2001-06

Effectiveness of Naval Surface Fire Support to the Army Brigade Commander in a Littoral Campaign

Ulloa, Juan K.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/2440

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

EFFECTIVENESS OF NAVAL SURFACE FIRE SUPPORT TO THE ARMY BRIGADE COMMANDER IN A LITTORAL CAMPAIGN

by

Juan K. Ulloa

June 2001

Thesis Advisor:	Eugene P. Paulo
Second Reader:	Arnold H. Buss

EFFECTIVENESS OF NAVAL SURFACE FIRE SUPPORT TO THE ARMY BRIGADE COMMANDER IN A LITTORAL CAMPAIGN

Juan K. Ulloa–Major, United States Army
B.S., United States Military Academy, 1990
Master of Science in Operations Research–June 2001
Advisor: Eugene P. Paulo, Department of Operations Research
Second Reader: Arnold H. Buss, Department of Operations Research

Since the end of the Cold War, the Army has been engaged in an unprecedented number of joint contingency operations that run the gamut from humanitarian efforts in Cuba and Haiti to peace-enforcing and peace-keeping in Bosnia to full scale war in Southwest Asia. These operations, the result of an increasingly complex international security environment, hint at future missions involving American forces aimed at protecting U.S. interests worldwide.

To engage and defeat future threats to our national security, the Army must transform itself into a more strategically responsive, lethal force. The Army is faced with the challenge of lightening the force while simultaneously increasing its survivability and lethality. Reach-back technologies from sea, air, and space can provide Army units with added lethality without encumbering them further.

This thesis analyzes the ability of the Army to effectively utilize Naval Surface Fire Support (NSFS) to provide indirect fire in support of brigade-sized units. The Fire Support Simulation Tool (FSST) takes the capabilities and limitations of weapon systems being studied and simulates their employment in the context of a well-defined scenario for analysis. The output from the simulation provides the input for the analysis of NSFS.

By comparing the utility of several well-constructed courses of action, the FSST can help decision-makers determine the effectiveness of NSFS within the context of the scenario being considered. The results of this analysis determined that although a myriad of issues such as training, mistrust, and synchronization must be addressed to make reach-back fires successful, there is strong *quantitative* and *analytical* evidence to support the effectiveness of NSFS to an Army Brigade commander engaged in a littoral campaign.

DoD KEY TECHNOLOGY AREA: Battlespace Environments, Command, Control, and Communications, Computing and Software, Conventional Weapons, Modeling and Simulation.

KEYWORDS: NSFS, simulation, Java, FCS, IBCT, Artillery, DD21, DD-21, Paladin, Crusader.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Effectiveness of Naval Surface Fire Support to the Army Brigade Commander in a Littoral Campaign	**5. FUNDING NUMBERS**
6. AUTHOR(S) Major Juan K. Ulloa	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

Since the end of the Cold War, the Army has been engaged in an unprecedented number of joint contingency operations that run the gamut from humanitarian efforts in Cuba and Haiti to peace-enforcing and peace-keeping in Bosnia to full scale war in Southwest Asia. These operations, the result of an increasingly complex international security environment, hint at future missions involving American forces aimed at protecting U.S. interests worldwide.

To engage and defeat future threats to our national security, the Army must transform itself into a more strategically responsive, lethal force. The Army is faced with the challenge of lightening the force while simultaneously increasing its survivability and lethality. Reach-back technologies from sea, air, and space can provide Army units with added lethality without encumbering them further.

This thesis analyzes the ability of the Army to effectively utilize Naval Surface Fire Support (NSFS) to provide indirect fire in support of brigade-sized units. The Fire Support Simulation Tool (FSST) takes the capabilities and limitations of weapon systems being studied and simulates their employment in the context of a well-defined scenario for analysis. The output from the simulation provides the input for the analysis of NSFS.

By comparing the utility of several well-constructed courses of action, the FSST can help decision-makers determine the effectiveness of NSFS within the context of the scenario being considered. The results of this analysis determined that although a myriad of issues such as training, mistrust, and synchronization must be addressed to make reach-back fires successful, there is strong *quantitative* and *analytical* evidence to support the effectiveness of NSFS to an Army Brigade commander engaged in a littoral campaign.

14. SUBJECT TERMS NSFS, Simulation, Java, FCS, IBCT, Artillery, DD21, DD-21, Paladin, Crusader, Combat Modeling, Field Artillery, Naval Gunfire.	**15. NUMBER OF PAGES**
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

EFFECTIVENESS OF NAVAL SURFACE FIRE SUPPORT TO THE ARMY BRIGADE COMMANDER IN A LITTORAL CAMPAIGN

Juan K. Ulloa
Major, United States Army
B.S., United States Military Academy, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 2001

Author: ________________________________
Juan K. Ulloa

Approved by: ________________________________
Eugene P. Paulo, Thesis Advisor

____
Arnold H. Buss, Second Reader

____
James N. Eagle, Chairman
Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Since the end of the Cold War, the Army has been engaged in an unprecedented number of joint contingency operations that run the gamut from humanitarian efforts in Cuba and Haiti to peace-enforcing and peace-keeping in Bosnia to full scale war in Southwest Asia. These operations, the result of an increasingly complex international security environment, hint at future missions involving American forces aimed at protecting U.S. interests worldwide.

To engage and defeat future threats to our national security, the Army must transform itself into a more strategically responsive, lethal force. The Army is faced with the challenge of lightening the force while simultaneously increasing its survivability and lethality. Reach-back technologies from sea, air, and space can provide Army units with added lethality without encumbering them further.

This thesis analyzes the ability of the Army to effectively utilize Naval Surface Fire Support (NSFS) to provide indirect fire in support of brigade-sized units. The Fire Support Simulation Tool (FSST) takes the capabilities and limitations of weapon systems being studied and simulates their employment in the context of a well-defined scenario for analysis. The output from the simulation provides the input for the analysis of NSFS.

By comparing the utility of several well-constructed courses of action, the FSST can help decision-makers determine the effectiveness of NSFS within the context of the scenario being considered. The results of this analysis determined that although a myriad of issues such as training, mistrust, and synchronization must be addressed to make reach-back fires successful, there is strong *quantitative* and *analytical* evidence to support the effectiveness of NSFS to an Army Brigade commander engaged in a littoral campaign.

DISCLAIMER

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF SYMBOLS, ACRONYMS, AND/OR ABBREVIATIONS

AO	Area of Operations
APC	Armored Personnel Carrier
APL	Applied Physics Lab
CA	Coverage Area
CD	Collateral Damage
COA	Course of Action
FA	Field Artillery
FM	Fire Mission
FM	Field Manual
FMT	Fire Mission Time
FSST	Fire Support Simulation Tool
GUI	Graphical User interface
IDI	Infantry Dug In
IIO	Infantry in the Open
JHUAPL	Johns Hopkins University Applied Physics Lab
LSV	Light Skinned Vehicle
MAUT	MultiAttribute Utility Theory
MOE	Measure of Effectiveness
NAS	Number of Available Shooters
NPS	Naval Postgraduate School
NSFS	Naval Surface Fire Support
PED	Probable Error in Deflection
PER	Probable Error in Range
PF	Percent Fired
PS	Percent Successful
TOF	Time of Flight
TPS	Total Percent Successful
US	United States

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to thank the following individuals and organizations for their help and support:

I would like to thank the Johns Hopkins University Applied Physics Lab for sponsoring my thesis research. Specifically, I would like to thank Jack Keane and Ted Smyth who were instrumental in setting up and executing a wonderful thesis tour. I would also like to thank Stephen Orloff and Vinnie Broderick who took me under their broad wings while at JHUAPL.

I want to also thank Lieutenant Colonel Eugene P. Paulo for his time and patience as my thesis advisor. His guidance, mentorship, and wisdom throughout the development of this thesis were absolutely critical. His encouragement is the reason this thesis even exists.

I also would like to extend a special thanks to my second reader, Assistant Professor Arnold H. Buss. His class on simulation and his part in the creation of SIMKIT sparked the idea behind creating the Fire Support Simulation Tool.

I would like to thank my mother and father for encouraging me to aspire to learn and to have an open mind. I love you both!

Finally, I thank my wife, Heidi, and my children – Madeline, Juan, and Isabelle. You are God's blessing to me and I love you all very much!

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Since the end of the Cold War, the Army has been engaged in an unprecedented number of joint contingency operations that run the gamut from humanitarian efforts in Cuba and Haiti to peace-enforcing and peace-keeping in Bosnia to full scale war in Southwest Asia. Over the last ten years, there has been a steady increase in rapid-deployment, multi-dimensional, joint contingency missions to combat these threats. As we move into the 21st century these missions will certainly become more and more complex and more commonplace. To meet the increased requirements to engage and defeat new threats to our national security, the Army must transform itself into a more strategically responsive, lethal force that is dominant across a broad spectrum of military operations such as peace-keeping, combating criminal and terrorist activities, and full scale war. The Army must fully develop its capabilities as an integrated, joint force able to synchronize the lethal and non-lethal fires of all services at the brigade level. Reach-back technologies from sea, air, and space can provide Army units with added lethality without encumbering them further. This thesis analyzes the effectiveness of Naval Surface Fire Support (NSFS) for use in supporting land-based Army forces in the littoral. Rather than simply analyzing the different characteristics and specifications of available indirect fire weapon systems to determine their effectiveness, a model was created. This model, the Fire Support Simulation Tool (FSST), takes the capabilities and limitations of the weapon systems being studied and simulates their employment in the context of a well-defined scenario. This thesis analyzes the data produced by the FSST to draw some broad conclusions about the future of indirect fire support for Army operations.

The objective of this thesis is to determine the effectiveness of NSFS to the Army at brigade level and below in a littoral campaign. The objective was further defined in terms of seven measures of effectiveness or MOE's – average fire mission time, percentage of missions fired, percentage of fired missions that were successful, percentage of total missions that were successful, percentage of the area of operations that were covered by indirect fires, number of rounds that caused collateral damage, and the average number of firing platforms available to fire other missions or mass fires.

Several courses of action (COA's) were created, each modeling a different task organization of indirect fires to support an Army maneuver brigade. The first COA includes three Army artillery batteries, the second has three naval surface support ships, the third has six Army artillery batteries, and the fourth has three Army artillery batteries and three ships. Each of these COA's was compared in two separate scenarios. The first scenario models fires in support of the Army Interim Brigade Combat Team (IBCT) during the year 2005, and the second models fires in support of the Army Future Combat Systems (FCS) during the year 2015.

The simulation accounted for the environment and the specific characteristics of the weapon systems in each COA to create data for each simulation run. By weighting the importance of each MOE, multi-attribute utility theory was used to combine the data for a COA into a single measure of its effectiveness. Since each simulation run was governed by a large number of random events, replicating a specific COA and scenario gave a range of values for each of the MOE's. This in turn yielded a range of values for the effectiveness of each COA. The final distribution of scores of the COA's was then compared to determine which one was the best for each scenario.

The results of the simulation show that by combining the strengths of Army artillery and naval gunfire, an Army Brigade commander can organize a fire support team that is better able to support his missions than a pure strategy. Although effectiveness is strongly dependent on the weighting scheme the commander adopts for the MOE's, this thesis has demonstrated that there can be an added benefit to using NSFS that should be explored further.

Although this thesis provides evidence supporting the use of NSFS in a support of Army operations in the littorals, there are a myriad of issues such as training, mistrust, and synchronization that must be addressed to make these types of joint campaigns successful. In the final analysis, it was determine that there is strong *quantitative* and *analytical* evidence to support the effectiveness of NSFS to an Army Brigade commander engaged in a littoral campaign.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Since the end of the Cold War, the Army has been engaged in an unprecedented number of joint contingency operations that run the gamut from humanitarian efforts in Cuba and Haiti to peace-enforcing and peace-keeping in Bosnia to full scale war in Southwest Asia.

> With the emergence of an increasingly complex international security environment, sources of conflict and tension are increasing. Sources of unrest and conflict range from competition between states to the instability caused by the collapse of states unable to meet the strains of resource scarcity, population growth, and ethnic and religious militarism. The technology enabling real-time transmission of information from any point on the globe has facilitated the rise of sub-national and transnational groups, including criminal and terrorist elements that may pursue objectives that threaten U.S. interests (Shinseki, Statement to 106th Congress, p. 3).

Over the last ten years, there has been a steady increase in rapid-deployment, multi-dimensional, joint contingency missions to combat these threats. As we move into the 21st century these missions will certainly become more and more complex and more commonplace. To meet the requirement to engage and defeat these threats to our national security, the Army must transform itself into a more strategically responsive, lethal force. This force must be dominant across a broad spectrum of military operations ranging from peace-keeping and humanitarian missions, to missions combating criminal and terrorist activities, to winning the nations wars on a military budget that continues to shrink and be scrutinized by public opinion. General Shinseki’s vision for the Army is one of transformation from a Legacy Force designed to defeat Soviet forces in Europe to an Objective Force designed to preempt and if necessary defeat threats from all corners of the globe. This transformation will ultimately result in a force designed to take advantage of technology to facilitate the ability to rapidly deploy forces and to synchronize and integrate combat power through the design of compatible systems throughout the Army, Navy, Air Force, and the Marines.

Based on General Shinseki's vision, the Army must be capable of deploying one Brigade anywhere in the world within 96 hours, have an entire Division on the ground within 120 hours, and have five Divisions on the ground within 30 days. This mission leaves the Army with the challenge of lightening the force while simultaneously increasing its survivability and lethality. To do that, the Army must break free from the paradigm of a self-sufficient and self-contained force. It must fully develop its capabilities as an integrated, joint force able to synchronize the lethal and non-lethal fires of all services at the brigade level.

This thesis analyzes the effectiveness of Naval Surface Fire Support (NSFS) for use in supporting land-based Army forces in the littoral. Rather than simply analyzing the different characteristics and specifications of available indirect fire weapon systems to determine their effectiveness, a model was created. This model, the Fire Support Simulation Tool (FSST), takes the capabilities and limitations of the weapon systems being studied and simulates their employment in the context of a well-defined scenario. This thesis analyzes the data produced by the FSST to draw some broad conclusions about the future of indirect fire support for Army operations.

A. BACKGROUND

Mastering the employment of naval gunfire is a documented part of the training of all Army forward observers and Field Artillery officers. Currently, however, the mechanism for synchronizing and integrating naval gunfire into Army operations is vaguely defined and is a process of trial and error that could take hours or days to adequately refine. Although the technology to effectively synchronize naval gunfire with Army operations in the littorals has existed for quite some time, there has never been a pressing need for the Army to rely on naval gunfire to support its operations. In the past, once deployed and fighting on the ground, Army forces have been relatively self-sufficient, with additional fires being considered a bonus rather than a necessity. However, in order to quickly deploy a brigade sized force that is both survivable and lethal in the most austere of environments characterized by a nonlinear battlefield, the Army must change not only the structure of its force, but also the mindset of its leaders. We must admit that we need the support of our sister services to accomplish not only our missions, but also the missions of our country. As we move forward into the 21st century and the Army transforms from a Legacy to an Objective Force, the need for additional lethal and non-lethal fires that are quick and responsive is paramount. We must design the capability to allow the near simultaneous engagement of the enemy anywhere he can be seen and engaged with precision fires from all applicable mediums of land, air, space, and sea. Those fires must be quickly and easily integrated into a seamless joint operation that for example might include Army forces on the ground with EW support from space, close air support from the Air Force, and air defense and NSFS in the littorals from the Navy.

On 06 March 2000, the Navy took a huge leap in the right direction by tasking the Johns Hopkins Applied Physics Lab (JHAPL) with identifying and refining potential opportunities "…for NSFS asset employment in support of Army Future Combat System Objective Force operations and force employment" (JHU/APL Task Statement, p.1). Through the coupling of NSFS capabilities and limitations with Army requirements for the Objective Force, JHAPL is analyzing the contributions the Navy can make to Army operations now and in the future as it designs, outfits, and fields future systems.

B. THE ARMY OF THE FUTURE – BREAKING THE PARADIGM

As we entered the Cold War in the years following World War II, the Army prepared for battle against the monolithic threat from the Soviet Union and her allies. That force was designed to fight and defeat a large armored enemy on a linear battlefield in Europe. This basic force structure and design eventually lead to the Army we know today as the Legacy Force.

The battles of the future will *not* be linear. They will consist of multi-dimensioned battlefields against enemies that could include criminals, terrorists, and dictators in obscure locations. The enemy will practice guerrilla warfare, he will have weapons of mass destruction, and he will exercise information warfare with impunity. He will be a student of our history. He will know that we lack commitment to long, drawn out conflicts, that we are averse to a high casualty rate, and that we are sensitive to international opinion and hence have a fear of collateral damage. He will know our strengths and weaknesses and work diligently to exploit them and defeat us. We must meet and defeat this *new* enemy threat by changing our tactics.

The fact that battlefields of the future will be multi-dimensional and nonlinear, emphasizes the necessity to develop a capability to exploit knowledge of the situation to preempt enemy actions. We must develop the capability to be a full spectrum force that is agile, flexible, and adaptive. This force must balance precision strike with precision maneuver, as well as deploy quickly and sustain itself logistically for long periods of time. We have to be able to integrate *all* combined, joint, and other available assets quickly and easily into a synchronized plan. We must develop the full dimension operational capability to rapidly and simultaneously hit the enemy from all angles and sustain that pressure until the enemy capitulates. To do this, we must free ourselves from the paradigm of a cumbersome, hard to deploy, heavily-armored force designed to do battle on the steppes of Russia and the deserts of the Middle East.

Our *new* force must exceed the lethality of our *old* force through overwhelming firepower gained by synchronizing and using assets from space, air, and sea that are not part of its internal structure. And, our *new* force must exceed the survivability of our *old* force by its ability to move quickly and hide when necessary, by having a short logistics trail, by exercising tactical and operational mobility far in excess of what the enemy is capable of. Future and developing technology will make this new force unmatched on the battlefield*, Soldiers on Point for the Nation, Persuasive in Peace, Invincible in war.*

1. Interim Brigade Combat Team

The Interim Brigade Combat Team (IBCT) is a force being developed using today's technology to accomplish the objectives outlined in General Shinseki's vision for the transformation of the Army. By 2005 the Army will have the capability of putting one brigade in the air in 96 hours, one Division on the ground in 120 hours, and five Divisions on the ground in 30 days. The IBCT will be capable of accomplishing the intent behind the transformation of the Army to the Objective Force using today's technology. By 2005 the Army projects that at least five Divisions will have IBCT brigades. Not all Divisions in the Army will be outfitted using the IBCT platform, however; some will be transformed directly from their current configurations to the Objective Force. The IBCT is basically an interim solution to deployability shortcomings of the legacy force, and will serve as a test bed for the future technologies that will be used when the Objective Force is fielded.

2. Future Combat System – Objective Force

The FCS-OF is the force of the future and will be fielded by 2015. The weapon systems, communications architecture, digital systems, vehicles, etc for that force are being designed based on technology projected to be available in 2015. By designing all aspects of the Objective Force including the structure from the ground up, this force will be fully integrated internally as well as integrated with the other services and our allies to the maximum extent possible. This integration and cutting-edge technology will make the Objective Force the most lethal, survivable, and mobile force ever employed.

C. ARMY OPERATIONS

To accurately model the reality of combat, it is imperative that a broad base line of Army doctrine and the fundamentals of fire support are understood and accounted for. Knowing doctrine offers insight into what assumptions can be made and how they will affect the simulation. Since gaining a basic understanding of Army doctrine is so crucial to understanding what this thesis is attempting to measure and how, included are a few short, concise paragraphs on the doctrine and tactics behind Army operations.

Doctrine for all Army Operations are outlined in the Army Field Manual designated FM 100-5. Although that doctrine is updated slightly from time to time, "[f]undamental to operating successfully across the full range of military operations is an understanding of the Army's doctrinal foundations the principles of war and the tenets of Army operations" (FM 100-5, Chapter 2). Below is a brief definition of those timeless concepts, which will help the reader to understand the doctrine behind how the Army fights and wins.

1. Principles of War

> The nine principles of war provide general guidance for the conduct of war at the strategic, operational, and tactical levels. They are the enduring bedrock of Army doctrine. The US Army published its first discussion of the principles of war in a 1921 Army training regulation. The original principles adopted by the Army, although slightly revised, have withstood the test of time. Today's force-projection Army recognizes the following nine principles of war (FM 100-5, Chapter 2).

- **Objective.** Essentially every conflict has an ultimate objective. All military operations should be focused on attaining that objective. Actions and intermediate operations that do not contribute to attaining the ultimate objective should be avoided.

- **Offensive.** Offensive operations seize that initiative from the enemy. They make him react to us. They are the most effective and decisive actions that can be taken to obtain the objective.

- **Mass.** *Mass the effects of overwhelming combat power at the decisive place and time.*

> Synchronizing all the elements of combat power where they will have decisive effect on an enemy force in a short period of time is to achieve mass. To mass is to hit the enemy with a closed fist, not poke at him with fingers of an open hand. Mass must also be sustained so the effects have staying power. Thus, mass seeks to smash the enemy, not sting him. This results from the proper combination of combat power with the proper application of other principles of war. Massing effects, rather than concentrating forces, can enable numerically inferior forces to achieve decisive results, while limiting exposure to enemy fire (FM 100-5, Chapter 2).

- **Economy of Force.** The principle behind Economy of Force is to allocate the minimum amount of combat power to secondary, low threat, and/or low priority efforts. Effective use of this prinicple allows for the maximum massing of combat power where it is most needed.

- **Maneuver.** *Place the enemy in a position of disadvantage through the flexible application of combat power.*

> Maneuver is the movement of forces in relation to the enemy to gain positional advantage. Effective maneuver keeps the enemy off balance and protects the force. It is used to exploit successes, to preserve freedom of action, and to reduce vulnerability. It continually poses new problems for the enemy by rendering his actions ineffective, eventually leading to defeat. At all levels of war, successful application of maneuver requires agility of thought, plans, operations, and organizations. It requires designating and then shifting points of main effort and the considered application of the principles of mass and economy of force. At the operational level, maneuver is the means by which the commander determines where and when to fight by setting the terms of battle, declining battle, or acting to take advantage of tactical actions. Maneuver is dynamic warfare that rejects predictable patterns of operations (FM 100-5, Chapter 2).

- **Unity of Command.** For every effort or operation there should be one commander whose goal it is to synchronize all assets into an operation aimed at attaining the objective. Unity of command helps us mass our combat power at the decisive place and time.

- **Security.** Security of the force gives the commander flexibility by reducing the enemy's opportunity to surprise us and reducing our vulnerability.

- **Surprise.** *Strike the enemy at a time or place or in a manner for which he is unprepared.*

> Surprise can decisively shift the balance of combat power. By seeking surprise, forces can achieve success well out of proportion to the effort expended. Rapid advances in surveillance technology and mass communication make it increasingly difficult to mask or cloak large-scale marshaling or movement of personnel and equipment. The enemy need not be taken completely by surprise but only become aware too late to react effectively. Factors contributing to surprise include speed, effective intelligence, deception, application of unexpected combat power, operations security (OPSEC), and variations in tactics and methods of operation. Surprise can be in tempo, size of force, direction or location of main effort, and timing. Deception can aid the probability of achieving surprise (FM 100-5, Chapter 2).

- **Simplicity.** Prepare plans and orders in a manner that is clear and concise. Tired leaders and soldiers have a difficult time issuing and following complex, unclear plans. A mediocre but simple plan executed vigorously and flawlessly is superior to a perfect, complex plan whose execution is laden with mistakes and misunderstandings.

2. Tenets of Army Operations

> Whenever Army forces are called upon to fight, they fight to win. Army forces in combat seek to impose their will on the enemy; in operations other than war, they seek to alter conditions to achieve their purpose. Victory is the objective, no matter the mission. Nothing short of victory is acceptable. The Army's doctrine describes its approach to generating and applying forces and force at the strategic, operational, and tactical levels. The Army's success on and off the battlefield depends on its ability to operate in accordance with five basic tenets: *initiative, agility, depth, synchronization, and versatility.* A tenet is a basic truth held by an organization. The fundamental tenets of Army operations doctrine describe the characteristics of successful operations. All training and leadership doctrine and all combat, combat support, and combat service support doctrine derive directly from, and must support, the fundamental tenets. The US Army believes that its five basic tenets are essential to victory. In and of themselves they do not guarantee victory, but their absence makes it difficult and costly to achieve (FM 100-5, Chapter 2).

- **Initiative.** Initiative is gained by action. It implies an offensive mindset and is gained by creating options for your own forces while constraining the options of your foe. Initiative on the battlefield requires leaders to anticipate events at all levels so their forces can act and react quicker than the enemy. In the attack it means gaining the initiative through surprise and maintaining the initiative through a fast, aggressive tempo that keeps the enemy in a constant state of disorientation. In the defense it means quickly turning the tables and shifting to the offensive. Initiative requires decentralization of decision-making authority to the lowest practical level, while maintaining a synchronized effort. Initiative is our ability to control where we fight, when we fight, and under what conditions we fight.

- **Agility.** Agility is our ability to react quicker than the enemy. It gives us the ability to quickly concentrate our forces on enemy weaknesses. Agility is needed to acquire and retain the initiative.

- **Depth.** Depth is the extension of the dimensionality of operations. Considering the depth of the battlefield involves consideration of space, time, resources, information, and the objective. Attacking in depth involves disrupting and defeating the enemy in these areas simultaneously. Employing our forces in depth involves using all available assets maximally. "Depth allows commanders to sustain momentum and take advantage of all available resources to press the fight, attacking enemy forces and capabilities simultaneously throughout the battlefield. Momentum in the attack and elasticity in defense derive from depth" (FM 100-5, Chapter 2).

- **Synchronization.** Synchronization is arranging activities in time and space to mass combat power at the decisive point and time. Synchronization of the fight gives us the ability to attack the enemy at multiple points with multiple assets in depth, sequentially, and simultaneously.

- **Versatility.** A versatile Army can adapt to meet diverse mission requirements.

> It suggests that all military organizations must have the ability to organize in different combinations of units and the capacity to redeploy from one area or region to another without the loss of focus. Versatility is the result of well-led, well-trained, and well-equipped forces; high standards; and detailed planning. Versatility ensures that units can conduct many different kinds of operations, either sequentially or simultaneously, with the same degree of success (FM 100-5, Chapter 2).

3. Fire Support in the AirLand Battle

AirLand Battle encompasses the three aspects of an operation: the deep operation, close operation, and rear operation. Deep operations attack the enemy in the battlespace beyond the close fight. Deep operations shape the close fight. Forces that are in immediate contact with the enemy are involved in close operations. Generally close operations are considered to be the current fight at the Corps and lower levels. Rear operations are generally protection and sustainment operations in the rear areas of our force. They can be targets of enemy deep operations. All three of these aspects of the AirLand battle must be executed simultaneously by the maneuver commander in order to defeat the enemy.

Fire support assets in the AirLand battle include all indirect-fire weapons, armed aircraft, and other lethal and non-lethal means. It includes mortars, field artillery, naval gunfire, and air delivered weapons, which include all conventional, chemical, and tactical nuclear munitions. Nonlethal means include EW, illumination, and smoke. The purpose behind fire support is to support the scheme of maneuver, mass fires, and delay, disrupt, or destroy enemy forces in depth. Fire support destroys, neutralizes, and suppresses the enemy through synchronizing all fire support assets with the scheme of maneuver to accomplish the mission.

The field artillery is the principal means of fire support available to the maneuver commander, and is doctrinally responsible for the integration of all fire support assets in support of the maneuver commander’s plan. As the senior coordinator of all fire support assets, the fire support coordinator (FSCOORD) from the field artillery is responsible for the integration of all fire support means to support the maneuver commander in the close, deep, and rear fight.

(a) The Army Fire Support Team

(1) Forward Observers. Forward observers are considered the *eyes* of the artillery. They identify the target and determine the location of the target in a manner that can be pinpointed on a military map. Based on the information they have about the target (posture, speed, dispersion, etc.) and the environment the target is in (snow, swamp, open fields, etc.) the forward observer formulates a fire mission using the following three separate transmissions:

- Observer identification and warning order. In this transmission the observer identifies himself and lets the fire direction center (FDC) know what type of fire mission he is going to request (example: fire for effect).

- Target location. In this transmission the observer tells the FDC the location of the target.

- Description of target, method of engagement, and method of fire and control. Finally, the observer describes the target (example: enemy platoon in the open), designates a method of engagement (example: high angle, high explosive), and a method of fire and control (example, at my command).

(2) Fire Direction. The FDC is the *brains* of the artillery. The FDC takes this request for fire and formulates an actual fire mission based on the tactical situation, the location of the howitzers, and available munitions. He develops technical firing data (what direction and elevation to fire, which munitions, and what amount and type of propellant) for the howitzers. This technical firing data is passed to the howitzers.

(3) Howitzers. The howitzers are the brawn of the artillery. They take the fire mission data from the FDC, load that data into the howitzer, and fire the mission.

(b) Naval Gunfire

Calling in Naval gunfire is similar to calling in field artillery. The forward observer is still the eyes, but the fire direction and the gun systems are linked together as one and execute the mission as a unit. There are several challenges to calling in Naval gunfire. The format of the fire missions are different, the communications nets are different (HF versus FM), and the ballistic trajectories, types of munitions, etc can be very different. These differences create difficulties in calling for and adjusting naval gunfire, but for the purposes of this thesis most of these difficulties will be modeled using one or two simple parameters to be discussed later.

D. THE ARMY OF THE FUTURE AND NAVAL SURFACE FIRE SUPPORT

The Army has always been a self-sufficient force capable of sustained ground operations. Other than logistics requirements, the Army has also been capable of executing its missions independent of the other services once deployed. This capability is a desirable one, and was a necessity when the United States alongside NATO was poised for combat against the Soviet Union and the Warsaw Pact. In order to create a force that has the rapid deployment capability of the Objective Force, the Army must reduce its logistic footprint and maximize its ability to integrate and utilize fires from other sources. Reducing the logistic footprint is accomplished by the development of equipment that shares many common parts; the development of lighter, more lethal, multi-purpose munitions; and by reducing the size of the force without reducing its lethality.

Reducing the size of the force without diminishing its lethality is accomplished by cultivating and developing the ability for reach-back fires and effects. These fires and effects can include EW and ADA support from ships in the area, electronic support and laser munitions from space, deep and close strike support from the Air Force, Navy, and Marines, and fire support from adjacent Marine, Army, Naval assets, just to name a few. This thesis focuses on the effectiveness of reach-back fires from NSFS assets in range of Army operations in the littoral.

E. FIRE SUPPORT WEAPONS CAPABILITIES AND LIMITATIONS OVERVIEW

For simplicity the only assets analyzed and compared are field artillery and Naval guns in Battery mass missions. The effect of mortar fire is not included in the analysis. The data below include only what is needed to conduct the thesis.

1. Year 2005 (IBCT)

By year 2005 the Army expects to have five Divisions outfitted with IBCT brigades. This force will be designed with current off-the-shelf technology and will meet the CSA's intent of one brigade deployed in 96 hours, one Division on the ground in 120 hours, and five Divisions on the ground in 30 days.

The primary indirect fire weapon system used by the IBCT will be the Paladin howitzer. The Paladin is capable of firing up to eight rounds in one minute and can engage targets up to 30,000 meters away using a high explosive round. The howitzer and its ammunition support vehicle can carry 138 complete rounds, and there will be six Paladin howitzers and ammunition support vehicles in an artillery battery (Paladin, WebSite for Defence Industries).

The primary ship that will provide naval gunfire in support of Marine and Army units will be the DDG-51, a destroyer. The DDG-51 is currently being upgraded, and should be capable of engaging targets up to 63 nautical miles away or about 112,300 meters (Lisiewski and Whitmann).

2. Year 2015 (FCS-OF)

By the year 2015 the Army will have fielded the FCS-OF. The capabilities and limitations of the weapons systems available for the Objective Force are projections based on Army and Navy requirements statements for the Crusader howitzer and the DD-21.

The primary indirect fire weapon system organic to the Objective Force will be a futuristic howitzer designated as the Crusader throughout this analysis. The Crusader should be capable of firing ten rounds in one minute and engaging targets at ranges greater than 40,000 meters. The howitzer and its ammunition support vehicle should be able to carry around 160 complete rounds (Crusader, WebSite for Defence Industries).

The primary ship that will provide naval gunfire in support of Marine and Army units will be the DD-21. The DD-21 will be capable of engaging targets up to 100 nautical miles away (about 178,200 meters). The DD-21 is projected to have two guns per ship with a maximum firing rate of 24 rounds per minute and a magazine capacity of over 1200 rounds (Lisiewski and Whitmann).

F. THESIS STRUCTURE

Chapter II will describe the methodology used in identifying the measures of effectiveness, creating and implementing the courses of action (COA) and scenarios for the simulation, and developing the logic, assumptions, and interactions that drive the simulation. Chapter III will provide a detailed analysis of the output from each scenario and COA. Chapter IV will summarize the results of the simulation, outline conclusions and insights gained from the simulation, and provide recommendations for further research of this subject.

II. RESEARCH METHODOLOGY

A. OBJECTIVE

The objective of this thesis is to determine the effectiveness of NSFS to the Army at brigade level and below in a littoral campaign. In order to determine the effectiveness of NSFS, the objective is defined in terms of measures of effectiveness or MOE's. The process of determining the MOE's begins by defining the problem statement in several more concise sub-objectives, called top-level objectives. When the top-level objectives are met, the system being measured or analyzed is effective. The process continues by successively redefining each of the higher-level objectives with lower-level objectives that are needed to satisfy them. These objectives continue to be redefined until they are quantitative in nature. These quantitative objectives are the bottom-level objectives or MOE's (Armstrong, p. 4-3).

These objectives are assembled in a tree-like structure beginning with the problems statement at the top and ending with the MOE's at the bottom. The final product shows the methodology used to determine the MOE's. This is a clear and concise picture of the problem statement and the data needed to analyze the problem effectively.

B. DETERMINING THE MEASURES OF EFFECTIVENESS

Effectiveness of any fire support asset can be analyzed by considering its ability to satisfy the following four objectives: Reliability, Lethality, Flexibility, and Survivability. Those four objectives are lower-level objectives to the overall objective of measuring the effectiveness of NSFS.

While all four objectives are important, for the purposes of this study differences in survivability between systems can be assumed to be negligible and are therefore not included. Survivability of an asset is how robust that system is on the battlefield. It involves issues such as maintainability, ability to withstand harsh conditions, protection of the crew, and ability to hide from, evade, or withstand enemy indirect and direct fire systems. To measure the survivability of a particular indirect fire system may require a very detailed simulation that could measure and replicate the enemy's ability to attrit or reduce the effectiveness of the system, account for the effects of weather and other conditions that affect maintenance, and a host of other contributors and detractors from survivability. The focus of this thesis is on the effectiveness of indirect fire systems, specifically the capability of NSFS to engage enemy targets to standard for the Army. Although survivability cannot be discounted, we make the assumption that the survivability of the assets being compared is reasonably similar and can be mitigated to the point where they are negligible in this study. In short, survivability, although important is not the crux of this analysis, and should be considered separately.

Dissecting the remaining top-level objectives in turn gives a quantitative way of analyzing the effectiveness of NSFS. The top-level objectives for determining the effectiveness of NSFS are shown in Figure 2.1 below.

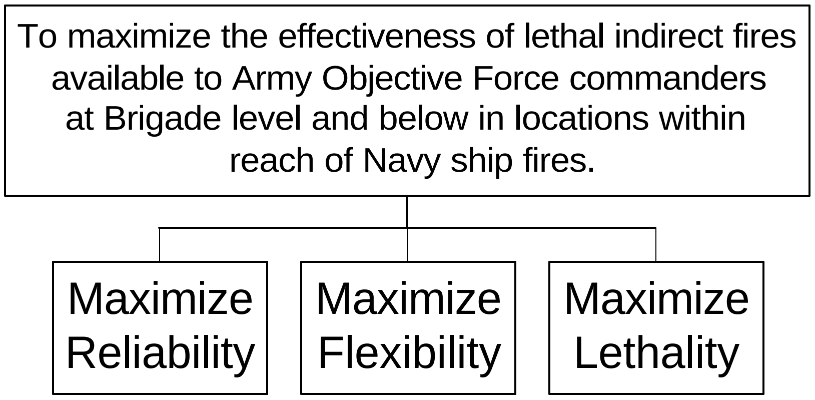

To maximize the effectiveness of lethal indirect fires available to Army Objective Force commanders at Brigade level and below in locations within reach of Navy ship fires.
Maximize Reliability
Maximize Flexibility
Maximize Lethality

Figure 2.1: Objectives Tree

1. Maximize Reliability

Reliability is an important part of integrating an asset into an operation. For an Army brigade commander to effectively integrate and synchronize the indirect fires into an operation, he must know that the asset will deliver its munitions on target at the prescribed place and time. The Reliability objectives tree below outlines the procedure and thought process behind determining the measurable, bottom-level tasks that determine the reliability of indirect fires.

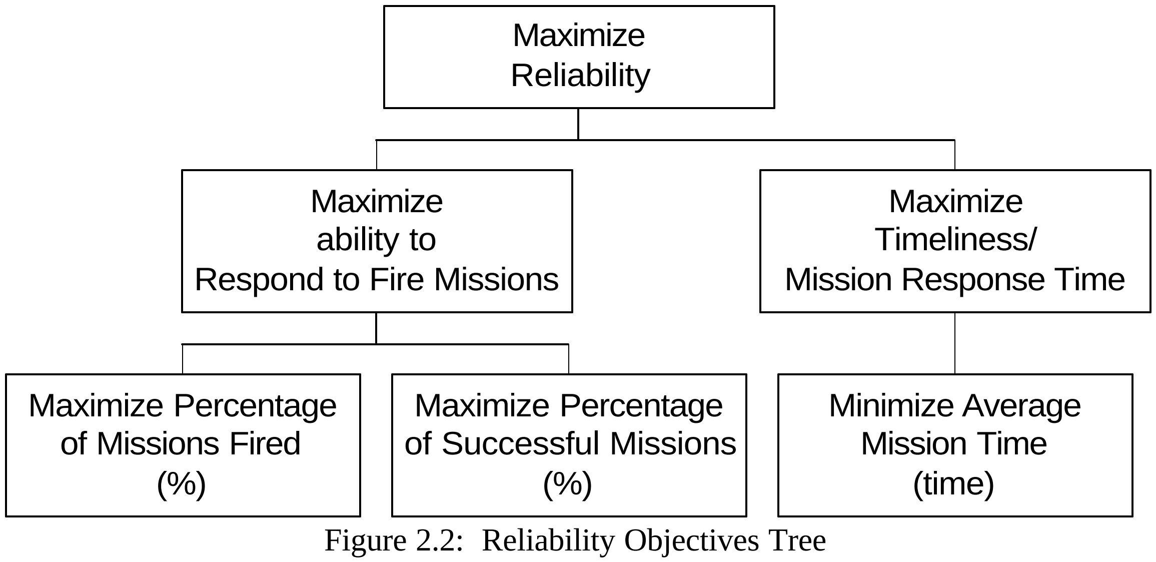

Maximize Reliability
Maximize ability to Respond to Fire Missions
Maximize Timeliness/ Mission Response Time
Maximize Percentage of Missions Fired (%)
Maximize Percentage of Successful Missions (%)
Minimize Average Mission Time (time)

Figure 2.2: Reliability Objectives Tree

To a brigade commander reliability of indirect fires generally consists of two screening criterion: (1) capability of an asset to respond to and engage various threats and

(2) timeliness in its response to the request for indirect fires. Quantifying each of these objectives results in the lowest level objectives needed to measure reliability. These lowest level measurable objectives are the percentage of missions that the asset was capable of engaging, the percentage of those missions that were successful, and the time that each asset took to engage each threat.

The percentage of missions that each COA *could* engage encompassed the brigade commander's first criteria for reliable indirect fires. This bottom-level objective tells the brigade commander whether the assets at his disposal can cover a sector of his tactical plan with the ammunition available.

The percentage of successful missions pertains to the second of the brigade commander's criterion. Measuring the percentage of successfully engaged targets of those that could be engaged tells the brigade commander how responsive the indirect fires are to his requests. It measures their ability to reliably engage the targets he wants engaged and inflict the requisite damage on those targets.

The average time to engage targets is a measure of the timeliness of the indirect fires in a particular COA. This pertains to the brigade commander's third criterion. The objective tree for reliability is shown in Figure 2.2.

2. Maximize Flexibility

Flexibility is the ability of an asset to perform and successfully accomplish diverse missions. Flexibility includes more than the ability to range targets. It includes having the right munitions to engage hardened targets and having the precision to engage targets that are positioned in awkward or protected locations. The three sub-objectives listed below were identified as crucial to measuring flexibility.

High precision allows an asset to be used to engage targets that are in close proximity to friendly troops or noncombatants. By measuring the number of errant rounds that induce collateral damage in each scenario, the simulation can measure the precision of indirect fires in that scenario. By carefully modeling this parameter, collateral damage for different environments can be measured. For example, in a rural setting, on average, collateral damage might only be induced by 0.1 percent of the rounds that are errant by 200 or more meters, while in an urban setting, collateral damage might occur with a 50% chance if a single round misses its mark by more than 50 meters. These specific parameters are included in the scenario.

Maximizing coverage allows one asset to provide indirect fires in the maximal number of situations. By measuring the percentage of the area of operations covered by indirect fires, we can compare the differences in different courses of action.

Ultimately, maximizing flexibility consists of the following two bottom-level, measureable objectives: number of rounds that cause collateral damage and percentage of the area of operations that is covered by indirect fires. The objectives tree for maximizing flexibility is shown below in Figure 2.3.

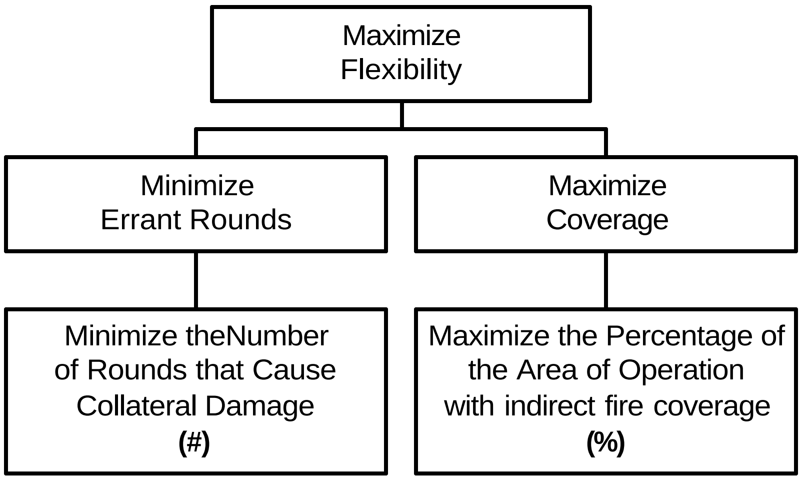

Maximize Flexibility
Minimize Errant Rounds
Maximize Coverage
Minimize theNumber of Rounds that Cause Collateral Damage (#)
Maximize the Percentage of the Area of Operation with indirect fire coverage (%)

Figure 2.3: Flexibility Objectives Tree

3. Maximize Lethality

Lethality is the cornerstone of Army operations. Without lethality, or the perception of lethality, we are ineffective. Precision and massing of indirect fires attains lethality.

By maximizing the availability of artillery at any given moment, we can measure the extent to which we can mass fires. It is worthwhile to note that although this objective is listed under lethality, it is really a multidimensional objective that gives the commander a measure of flexibility and reliability as well. The average availability of firing platforms indicates how likely it is that at any given moment he can effectively engage a target (reliability), and can serve as an indicator to the commander that he has the flexibility to shift assets and move assets on the battlefield to enhance his ability to engage the enemy. Ultimately, maximizing the availability of firing platforms provides the commander with the lethality he needs to mass indirect fires, the flexibility he needs to move assets on the battlefield, and the reliability he wants to immediately engage targets as they become available.

A single fire mission supports the overall mission of the organization by doing its part in the concept of the operation. By maximizing the total percentage of missions engaged successfully, the success of the mission is maximized. This differs from the reliability measurement, since it only measures the percentage of missions that result in success of the ones engaged (# success/# engaged * 100%), while reliability measures the percentage of targets successfully engage with respect to the total number that arrive (# success/total * 100%). The objectives tree for maximizing lethality is shown below in Figure 2.4.

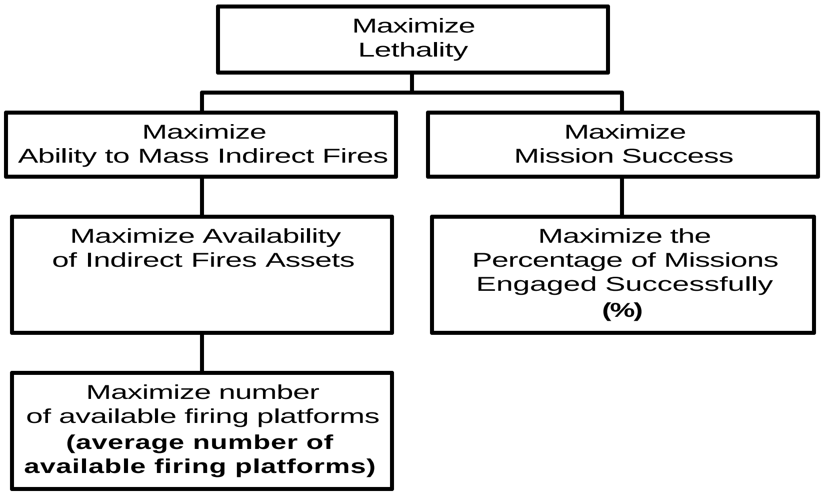

Maximize Lethality
Maximize Ability to Mass Indirect Fires
Maximize Mission Success
Maximize Availability of Indirect Fires Assets
Maximize the Percentage of Missions Engaged Successfully
(%)
Maximize number of available firing platforms
(average number of available firing platforms)

Figure 2.4: Lethality Objectives Tree

C. DECISION MATRIX

The top-level objectives of each objectives tree are those objectives that are most important for measuring the effectiveness of indirect fires to the brigade commander, while the bottom-level objectives provide a quantifiable and measurable method of comparing the performance of the top-level objective under different scenarios. These bottom-level objectives are measures of effectiveness (MOE's) for the top-level objectives. They are the quantitative data that help the decision-maker pick a COA.

To measure the effectiveness of each COA, multi-attribute utility theory or MAUT, was used. This method provides a simple, relatively intuitive way to weight and quantify the value of very different decision-making criterion. To use MAUT, two basic assumptions must be met:

> **1.** It must be possible for the decision maker to consider and judge the relative weight of any combination of factors. That is, it must be possible to consider not only the weight of factor 1 (F_1), but also the weight of both F_1 and F_2.
>
> **2.** Weights are assumed to be additive. That is, given the weight of F_1 and the weight of F_2, the weight of both F_1 and F_2 is the sum of their individual weights (Canada and Sullivan, p223).

If both of these assumptions hold, the decision maker can measure and compare the effectiveness of each COA in a given scenario using MAUT. First, the decision-maker or his representative weights each of the MOE's based on its relative importance. Each raw score for each MOE in a particular COA is then compared with the corresponding raw scores from each other COA. The "best" raw score is assigned a utility of 1, while the "worst" raw score is assigned a utility of 0. Utility of the remaining MOE's is assumed to be linear, and is computed using the following formula, where i = MOE number, j = COA number:

$$\text{Utility Score MOE}_{i,j} = (x_{i,j}\text{-worst}_{i,j})/(\text{best}_{i,j}\text{-worst}_{i,j}),\ \forall\ i,j$$

To get the utility score for $\text{MOE}_{1,1}$ (1st MOE for the 1st COA), the raw score of the $\text{worst}_{1,1}$ is first subtracted from $x_{1,1}$. This quantity is then divided by the raw score of the $\text{worst}_{1,1}$ subtracted from the $\text{best}_{1,1}$.

The total utility of each COA is computed using the following formula:

$$\sum_{i=1}^{n} MOE_{i,j} * WEIGHT_{i,j} \cdots \forall_{j}$$

The best COA is the COA with highest total utility score (Canada and Sullivan, p. 228).

D. SIMULATION – DEVELOPING A FIRE SUPPORT SIMULATION TOOL (FSST)

The Fire Support Simulation Tool (FSST) is a discrete-event simulation written in the programming language JAVA. The simulation uses Simkit, a discrete-event simulation package created by Assistant Professor Arnold H. Buss and LT Kurt Stork written in JAVA (Stork, 1996).

The objective of the FSST is to obtain the raw MOE data for each COA as determined by the value systems design described earlier. Although high-fidelity simulations exist that can provide the raw MOE data, they are cumbersome, complex simulations that take days, weeks, and even months to prepare and execute. Even then, these simulations do not package the output in a way that can be easily interpreted.

The objective of the FSST is to provide the user with a portable simulation that can be prepared and executed in minutes, is simple to use, and provides results in an easy-to-read and understand format using intuitive analysis techniques that anyone can understand. The intent of the simulation is to provide quick, broad insight into the advantages and disadvantages of different task organizations of indirect fire assets (each task organization represents a different COA), and to provide a quantitative measure of the effectiveness of that task organization. By investigating a well-selected sample of COA's the effectiveness of NSFS can be determined.

1. Why Simulation

Because of the complex, stochastic nature of this problem, there is no closed-form solution to measuring the effectiveness of NSFS. Because of this, simulation is good tool to investigate the effectiveness of NSFS using the MOE's outlined above (Law and Kelton, p. 6). By accounting for the stochastic nature of target arrival times and fire mission times, a simulation can draw a complete picture of the strengths and weaknesses of each COA in terms of the MOE's and their variances when applicable. This is powerful information for a decision-maker and offers valuable insight into the performance of the assets being evaluated. For example, a commander who is very concerned with success of critical fire missions might choose an asset that is more stable (has lower variance in success rate) over one that is more chaotic with a higher average success rate. This simulation should reveal those chaotic behaviors allowing the commander to make a more informed decision. Since this simulation is easy to set-up and execute, it should also allow any staff to quickly create and run multiple courses of action (COA's) for each scenario. The staff can then present the results and their analysis and their best scenarios to the decision-maker, expanding his flexibility and offering even more insight into the behavior of his assets in his environment.

2. Java and SIMKIT Overview

Java is an object-oriented language that is ideal for this simulation. By depicting processes, entities, and actions as objects, Java creates a tangible, intuitive, expandable, model of reality in its programs. This coupled with the fact that Java is a high-level, powerful, platform independent programming language makes it an ideal programming tool for this thesis. Simkit is a flexible simulation tool kit that allows the coupling of many independent objects into one complex simulation. Taken as whole, Simkit and Java offer unmatched flexibility and robustness for creating a simulation tool.

Simkit is a discrete-event simulation tool that uses *next-event time advance*. A discrete-event simulation models a system as it changes state over time. Next-event time advance simulations model only the changes in the environment, such as arrival of a target firing of a target, etc. (Law and Kelton, p. 6). By contrast, *fixed-increment time advance* simulations model the state of the system at each discrete time-step. For the purpose of this thesis, the discrete event simulation will be discussed within the context of the next-event time advance model. This method facilitates modeling the behavior of the fire support battle by maintaining a high level of time fidelity while reducing run times by fast-forwarding to the times in the COA that are eventful. For example, by running a time-step scenario of a peacekeeping mission, we might find that we execute a specific fire mission once every three months or 7,776,000 seconds plus the actual (insignificant) mission execution time. If we ran this simulation 1,000,000 times faster than real time, and we wanted mission times in seconds, it would take an average of 7.776 seconds per trial and 7,776 seconds or over two hours to execute 1000 trials to get confidence interval and variance data. By using an event-based simulation, if average mission times were ten minutes, we would only have to execute that ten minutes for each trial. This would take an average of 0.0006 seconds per trial and about 0.6 seconds to execute 1000 trials. While both methods are viable and work, one is clearly a better choice.

3. Queuing Theory

In general terms, queuing systems consist of one or more servers that provide a service to an arriving customer. Customers arrive at discrete times for service. When they find that all servers are busy they generally enter one or more queues in front of the server. Queuing models attempt to model the behavior of queuing systems by quantifying the interarrival and interservice times of the customers and the servers (Law and Kelton, p. 94).

The arrival and service of customers at a gas station fit the description of a system that can be modeled using queuing theory. At a gas station, customers arrive and are serviced in much the same manner that fire missions arrive and are serviced by available indirect fire assets. This analogy is surprisingly accurate, as will be explained below.

Consider for example that a car arrives at the service station. There are several *different* types of vehicles. For the purposes of a gas station there would probably be at least four different types: those that run on regular unleaded, unleaded plus, super unleaded, and diesel. Once we have decided what type of vehicle we have, we then look for a pump that provides that type of fuel. If one is available we fuel up. If not, we get into a line or queue.

Now, imagine fire missions as cars and indirect fire assets as the fuel pumps. Each fire mission has attributes such as arrival time, type (armor, infantry, etc.), location, and desired effects by the server (destroy, neutralize, or suppress). Each different type of mission is defined by it attributes, which are analogous to the different types of cars (diesel, unleaded, etc.).

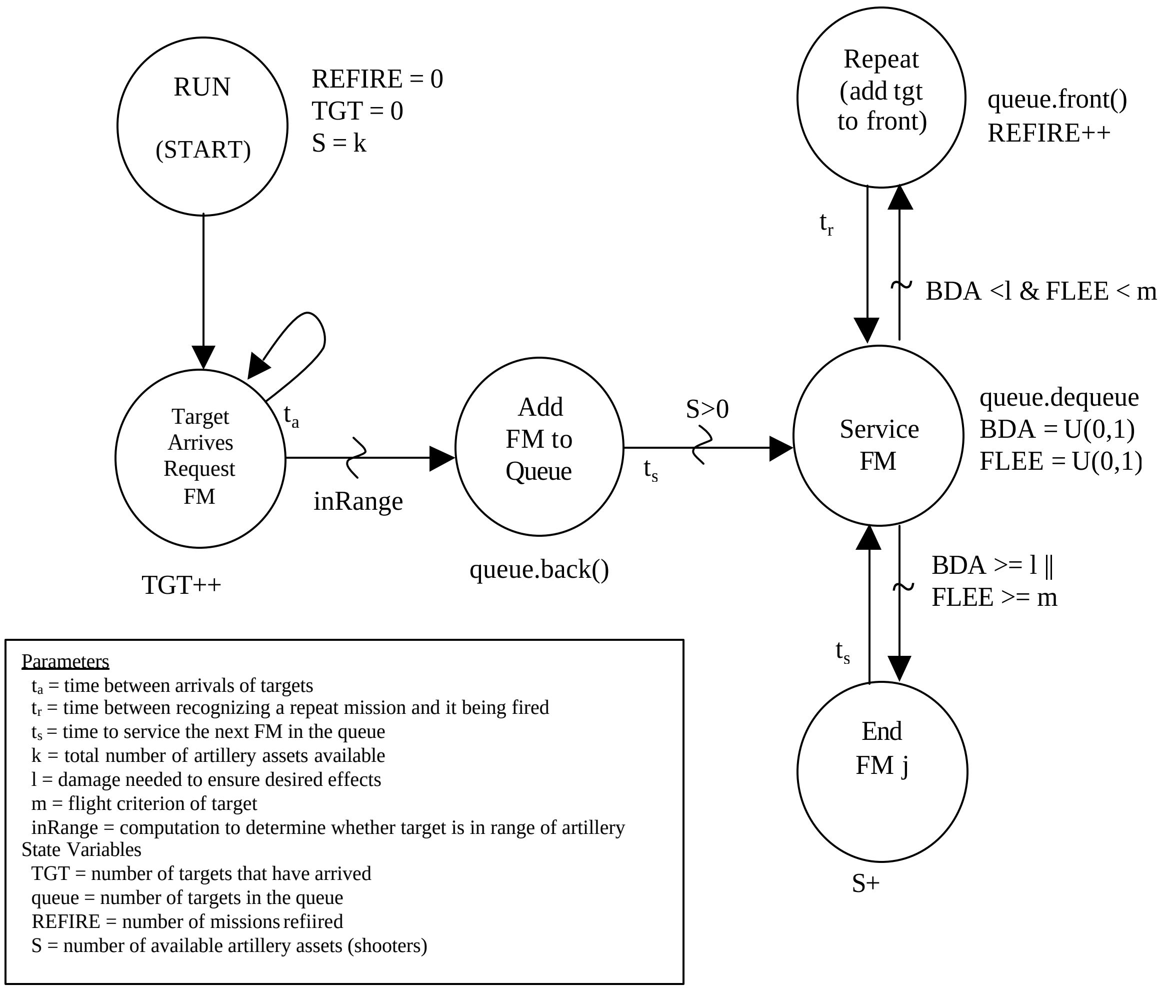
The servers are the indirect fire platforms whose mission it is to change the status of the target from its initial condition to one of destroyed, neutralized, or suppressed. Each server or shooter has a list of attributes that facilitate its ability to inflict the desired effects on the target. They include:

- the shooter’s location
- service time and distribution of each fire mission
- the shooter’s rate of fire
- lethality and accuracy of the round
- the number of gun tubes associated with the shooter
- the number of rounds the shooter has available

Each shooter is defined by its attributes, which are used to model the different types of indirect fire assets such as Paladin battalions, DDG-51's, or M198 batteries. For the purposes of this thesis, the FSST will assume only a few of the more common fire mission types and will limit the characteristics of the delivery systems to what is necessary to execute those fire missions.

4. Event Graph Development

The event graph shown in Figure 2.5 is a basic depiction of how the simulation works. The actual model is too complex to show in one simple event graph, but the basic model is depicted below in this one-dimensional event graph of a queuing model with one target or fire mission type and one server type.



Parameters
t_a = time between arrivals of targets
t_r = time between recognizing a repeat mission and it being fired
t_s = time to service the next FM in the queue
k = total number of artillery assets available
l = damage needed to ensure desired effects
m = flight criterion of target
inRange = computation to determine whether target is in range of artillery
State Variables
TGT = number of targets that have arrived
queue = number of targets in the queue
REFIRE = number of missions refiired
S = number of available artillery assets (shooters)

Figure 2.5: Event Graph

The circle with RUN sets up the queuing model by initializing all attributes. The simulation begins with the arrival of a target that causes the model to initiate the arrival of another target at some discrete time, t_a, in the future (circle with “Target Arrives, Request FM”). The interarrival time, t_a, of the targets is modeled by random exponential interarrival times. When a fire mission arrives, it is queued if it is within range of a shooter (circle with “Add FM to Queue”). The algorithm discussed above determines the shooter whose queue that fire mission goes into. If the shooter is immediately available, the fire mission is processed t_s time units later (circle with “Attack tgt”). The parameter t_s is a function of the range from the shooter, the shooter type, and other variables.

Once the target is attacked, if the desired results are achieved *or* the target has fled, the mission is ended (circle with “End FM”) and that shooter becomes available for another fire mission. If that particular shooter has another fire mission in the queue, it services that fire mission t_s time units later. If the desired results are *not* achieved *and* the target is still available, the fire mission is immediately repeated, with effects being considered t_r time units later. The parameter t_r is based on the time it takes for that particular shooter to refire the mission.

Although this event graph does not depict it, the FSST computes and maintains the number of arrivals, the number of missions rejected for any reason (such as range or lack of ammunition), and the number of successful and unsuccessful missions.

5. The Simulation

(a) Simulation Overview

Each arriving target is identified and engaged somewhere in a box that we consider the area of operations. To simulate different types of missions, units, and/or tactics the user can vary the size of that box. Throughout the battle, these targets would be identified and engaged in different areas of the box.

The distribution of the arriving targets within the box is scenario-based, and can be varied by the user. For instance, in a guerilla-type operation where there is no built-up enemy, and our forces are deployed in a decentralized manner, we might expect to acquire targets uniformly across the box since the enemy has freedom of maneuver and he is probably much more familiar with the terrain than we are. By contrast, in a conventional-type operation, we might expect to acquire targets uniformly across our front and exponentially in the depth of our position since we own the ground we are occupying and have well-defined boundaries that are protected. Again, to best model the scenario, the user can vary each of these.

The user will also be able to tactically place his artillery battalions and naval assets in the area of operations according to the scenario. Artillery and ships, once placed, will not move throughout the scenario. Figure 2.6 is a graphic depiction of a sample scenario with artillery and naval range fans depicting limits of engagement for these assets. The user will be able to easily set up, simulate, and analyze a scenario like this.

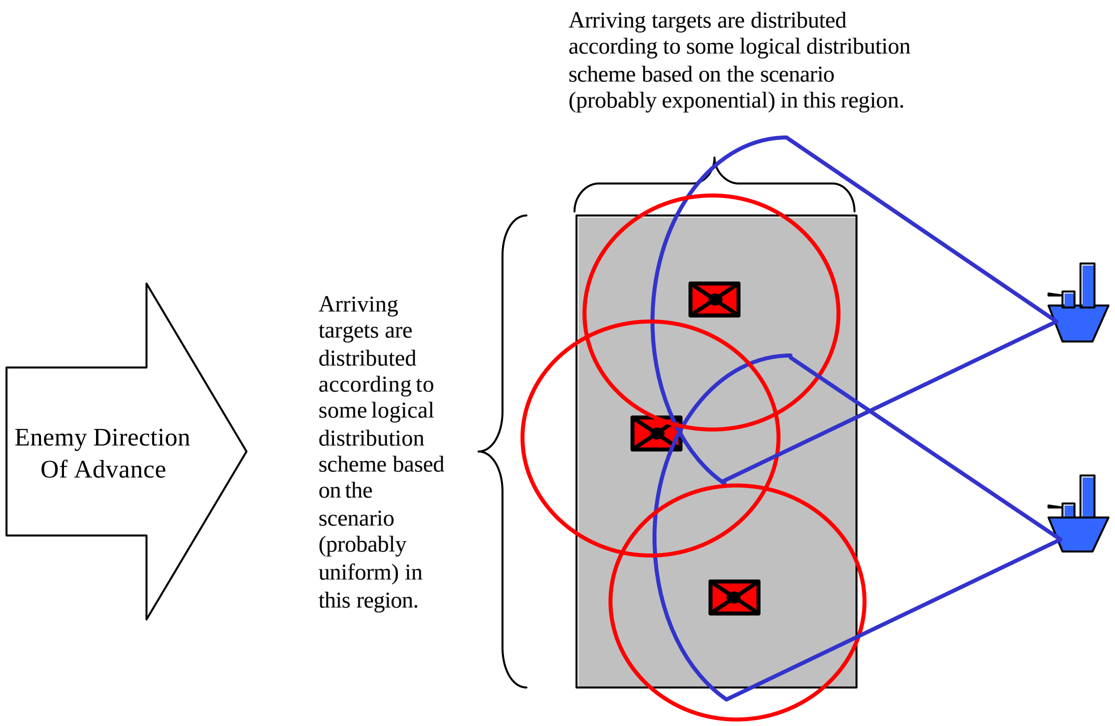

Arriving targets are distributed according to some logical distribution scheme based on the scenario (probably exponential) in this region.
Enemy Direction Of Advance
Arriving targets are distributed according to some logical distribution scheme based on the scenario (probably uniform) in this region.

Figure 2.6: Sample Scenario Graphic

Each scenario involves running the simulation for each COA listed above. By varying the inputs for the model such as parameters for the indirect fire assets and the properties of the area of operations (dimensions and distribution of arriving targets) the FSST can model the scenario in which each COA will occur.

(b) Target Arrivals

Targets arrive at a rate corresponding to a distribution. Based on past simulations, targets probably would arrive at an exponential rate with a mean based on the situation. This simulation allows the user to determine the arrival rate of the targets and the mean interarrival time (i.e. exponential with mean 5.0). Each target will then be further defined by its type – armor, armored personnel carriers, light skinned vehicles, infantry in the open, or infantry dug in, and the mission associated with the target, destroy, neutralize, or suppress. Again, the user can determine the percentages of each type of target and the mission associated with each target.

An example of how the arrival process works is as follows. A random number generator determines the arrival time of the first target. Another randomly generated number stochastically determines the target type. A third randomly generated number determines the mission associated with the target, and a random target location is generated based on the distribution of the locations of targets. The first target then enters the model, and another target arrives at a randomly generated interarrival time based on the arrival distribution, and the process begins again. The distribution of target types and mission types used for this thesis are shown below in tables 2.1 and 2.2.

Mission Types	**Percentage**
Destroy	30%
Neutralize	50%
Suppress	20%

Table 2.1: Distribution of Mission Types

Target Types	**Percentages**
Armor	40%
Infantry in the Open	10%
Infantry Dug In	0%
Armored Personnel Carrier	30%
Light Skinned Vehicle	20%

Table 2.2: Distribution of Target Types

(c) Target Servicing

When a target enters the model, it becomes a fire mission and is sent to a specific artillery or naval gunfire unit called a shooter. Each shooter is queried to determine whether it can range the particular target and whether it has the ammunition needed to engage the target. Once it has been determined which assets can effectively engage the target, an asset is chosen based on a weighting of the following criteria – platform or shooter type (NSFS or field artillery), number of fire missions in that shooter's queue, probable error in range, and shooter to target range.

A version of MAUT is used for the selection of a shooter for each target. The value of each of the first three criteria for a particular shooter is compared with the corresponding values of all shooters that can effectively engage the target. Subscripting the criteria being compared using the letter j, the utility of each shooter is computed with respect to each criteria using the following formula:

$$\text{Utility} = (x_j\text{-worst}_j)/(\text{best}_j\text{-worst}_j)$$

The total utility for each shooter is then determined using the following formula:

$$\left[\sum_{i=1}^{3} Utility_i * WEIGHT_i\right] + WEIGHT_{NSFS} + WEIGHT_{FA}$$

The shooter that gets the fire mission is the one with the highest total utility score. If no assets have the required number of rounds to effectively engage the target, the asset with the most rounds that can range the target is chosen. The fire mission is then put into the shooter's queue. Figure 2.7 below is a flow chart depicting how a specific shooter is selected.

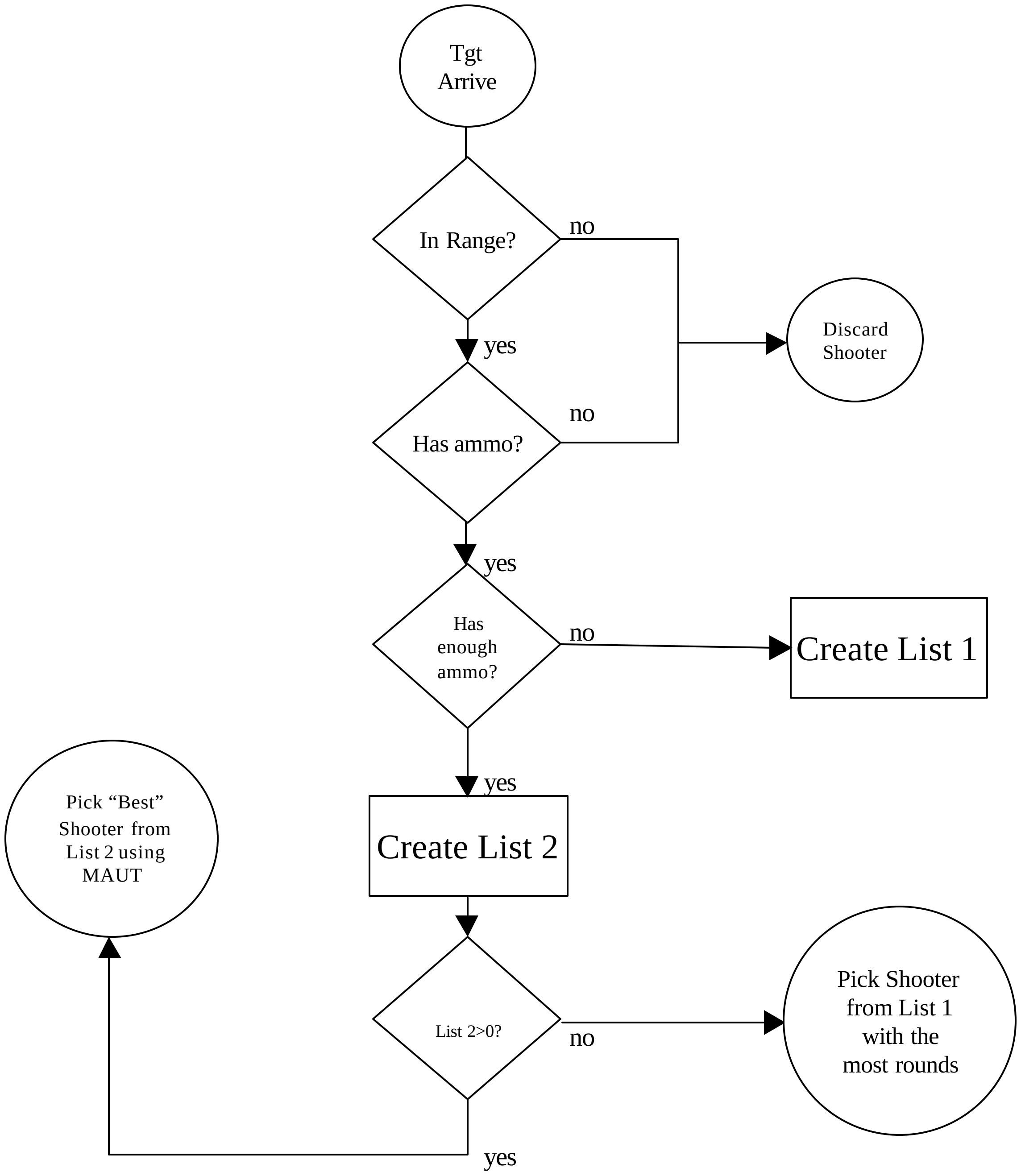


Figure 2.7: Fire Mission Engagement Process

The simulation tracks which assets can and do engage, the total time between the arrival and the execution of the fire mission, the number of successful, unsuccessful, and rejected fire missions, and the number of rounds that cause collateral damage. It can then, through simulation, assign values to each MOE under each COA.

(d) Target Engagement

Each shooter engages targets when they are at the front of the queue and no targets are being serviced. The processing time (t_p) and engagement times (t_e) are randomly determined from the distribution of the processing and engagement times entered for that shooter, and the time of flight (tof) for the rounds is computed based on the target range. By accounting for the rate of fire, the number of rounds desired, and the number of guns associated with the chosen shooter the total time for the fire mission can be computed using the following formula:

$$t_p + t_e + \text{tof} + \lceil (\#\text{rounds})/(\#\text{ tubes}) \rceil * (\text{rate of fire})$$

(e) Mission Success

The location where each round lands is determined stochastically using the probable error in range, probable error in deflection, and shooter-target range. The definition of success for a particular mission is pre-determined in the scenario, and is a function of the number of rounds that land within a certain distance of the target for a particular target type (i.e. armor) and mission type (i.e. destroy). For destroy and neutralize missions that distance is the burst radius of the round. For suppression missions, that distance is 2 times the burst radius since the objective is mainly to distract and rattle the enemy, not to kill him. To determine if the mission is successful, the simulation compares the number of *hits* needed with the number of *hits* the target has already sustained plus the number of additional *hits* if any. If the total number of *hits* sustained is greater than or equal to the number needed, the mission is successful.

If a mission is successful, the mission is ended, and the shooter fires the next mission in the queue, or waits for the next mission if none are currently queued. If the mission is not successful, the target "remembers" how many rounds have had the desired effects, and the mission is repeated if 1) the target has not fled – determined stochastically by scenario intput and 2) if the shooter still has rounds available. If both conditions are met, the mission is repeated. If not the mission is ended and is unsuccessful. The engagement time for repeated missions is generally faster than for the initial volley. The assumption is that the guns are already trained on the target, and are awaiting repeat or end of mission orders. The following formula determines the time for repeating the mission:

$$\text{tof} + \lceil (\#\text{rounds})/(\#\ \text{tubes}) \rceil \,/\, (\text{rate of fire})$$

This process is repeated until either the mission is successful, the target flees, or the shooter runs out of ammunition. The total mission time is then computed and tallied. Successful and unsuccessful missions are also tallied.

(f) Rejected Missions

If a target cannot be engaged because there are no shooters available with any ammunition, that mission is rejected. Rejected missions are tallied during the simulation.

(g) When a Shooter Runs Out of Ammunition

If a shooter runs out of ammunition, the current mission being fired ends successfully or unsuccessfully as outlined above, and all missions in that shooter’s queue are rejected.

(h) Collateral Damage

Collateral damage can occur each time that a target is engaged. If a round misses its intended target by more than a distance predetermined in the scenario, that round *can* cause collateral damage. Each errant round causes a random number to be generated, which determines stochastically whether that round causes collateral damage according to a predetermined percentage of errant rounds that cause collateral damage (a scenario input).

(i) Coverage Area

The area of the box that the course of action being simulated can cover is known as the coverage area. The closer the number is to one, the better the COA is with respect to that particular MOE. In order to determine the coverage area, the simulation is run for the particular COA using 100,000 target arrivals to get as accurate a measurement as possible. Additionally, each shooter is given an infinite number of rounds and the targets arrive at a location uniformly distributed within the box. The coverage area can then be measured by taking the number of missions accepted and dividing by the total number of missions. This number is then input into the COA being simulated as a parameter.

E. SCENARIO DEVELOPMENT

A scenario consists of the parameters that quantify enemy and friendly actions, the effects of the environment and terrain on the military operation, the level at which the battle is being executed, and the year in which the effectiveness of NSFS is being measured. These parameters are kept constant for each COA within the scenario so that COA's can be compared using a common criterion.

Terrain quantifying parameters include the size of the area of operations (AO), the definition of an errant round, and the probability of collateral damage by an errant round. Enemy parameters include the rate at which targets arrive, their location in the AO, and the target type (armor, infantry, etc.). Friendly parameters include the distribution of the mission types (i.e. destroy, neutralize, suppress), the attack criterion for different targets (i.e. how many rounds to fire in suppression of armor), and the definition of a successful mission in terms of how many target hits are required to get the desired effects. Additionally, the scenario includes the weighting scheme the decision-maker uses to choose an asset to engage targets, which will be explained in more detail later.

The level of the battle is the echelon at which the battle is being fought. Both scenarios for this thesis focus on providing indirect fire support for an Army brigade-level unit. The year that the scenario models does not determine parameters for the scenario in and of itself, rather it determines what technologies and systems are projected to be available, which in turn dictate the parameters of the artillery platforms to be used in each COA. This will be explained in more detail in the next section.

The terrain, enemy and friendly parameters, and level of the battle will remain basically the same for each of the following two scenarios being modeled. Additionally, since the objective of the thesis is to measure the effectiveness of NSFS to the Army brigade commander, the battle for both scenarios is limited to the brigade fight. The major differences between the two scenarios will be the year in which they occur.

1. Scenario 1 – IBCT

This scenario is called the IBCT scenario because it is centered upon the year 2005, and therefore limits the COA's to equipment that is either available now or is projected to be available by year 2005. For the field artillery, this means the M109 Paladin is available with a maximum firing range of 30,000 meters. For the naval gunfire this means using DDG-51's with a maximum range of 63 nautical miles (about 112,300 meters).

2. Scenario 2 – FCS-OF

The FCS-OF scenario models the effectiveness of indirect fire systems that are projected to be available by year 2015. For the field artillery, this means the Crusader is available with a maximum firing range of about 45,000 meters. For the naval gunfire this means using DD-21 with guns having a maximum range of about 100 nautical miles (about 162,000 meters).

F. COA DEVELOPMENT

Each COA is developed within the context of a specific scenario that determines the year and the tactical and operational considerations of the battle. The year determines the firing platforms available for the COA.

Each COA developed with the scenario includes which firing platforms are being used, their technical capabilities and limitations, and their locations on the battlefield. The technical capabilities and limitations of the firing platforms include maximum range, firing rate, probable error in range and deflection, distributions for target processing and preparation for firing, and the bursting radius of a single round.

The overall objective of the experiment is to measure the effectiveness of NSFS within the context of each scenario. Carefully designing three or four COA's in each scenario helps to isolate the effect being measured. Tables 2.3 and 2.4 below show an overview of the COA's and the factors that are varied or changed for each COA.

FACTORS	**COA (same for each Scenario)**			
	A	**B**	**C**	**D**
Field Artillery	3	0	6	3
Naval Ships	0	3	0	3

Table 2.3: COA Overview

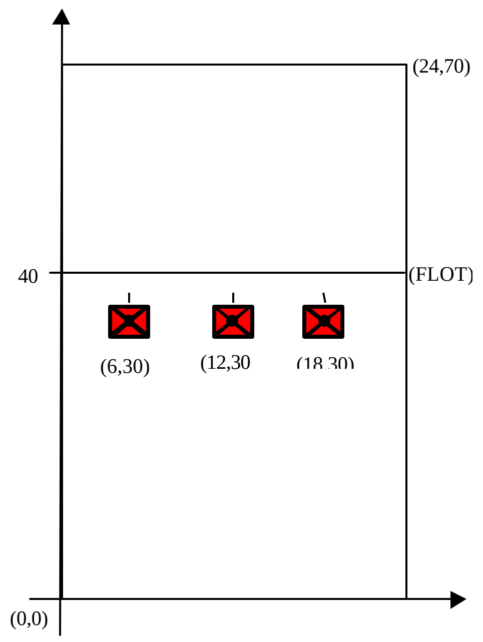
Parameter	**IBCT**		**FCS-OF**	
	FA	**NSFS**	**FA**	**NSFS**
Number Guns	6	2	4	2
Rounds per Ship/Battery	2520	2400	2520	2400
Range (1000's meters)	30	112	45	162
Munition Burst Radius	50	75	50	75
PER	35	100	1	10
PED	2	2	1	2
Max Rate of Fire	8	24	12	24
Acquisition Time Distribution	Normal	Normal	Normal	Normal
Mean Acquisition Time	4.0	5.0	2.0	2.5
Acquisition Std Deviation	0.75	1.0	0.25	0.5
Firing Time Distribution	Normal	Normal	Normal	Normal
Mean Firing Time	1.0	0.5	0.25	0.5
Firing Time Std Deviation	0.2	0.1	0.05	0.1

Table 2.4: Hardware Parameter Overview

1. Organic Army Fire Support Only

This COA is the base-line model for the effectiveness of Army indirect fires. It involves running the simulation with only one organic Field Artillery battalion in direct support of a maneuver brigade.

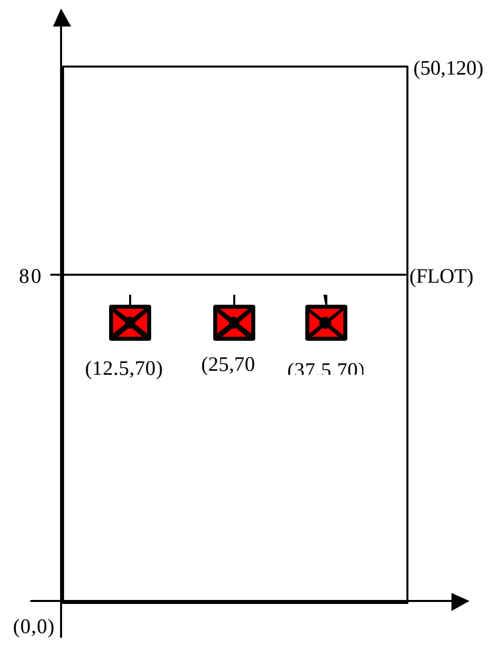
(a) IBCT



```
The AO properties
X distribution = Uniform(0,24)
Y distribution = Exponential(50)
Arrival distribution = Exponential(2.5)

Shooter properties
  meanProjectileVelocity = 70000 m/min
  Paladin Battery:
    acquireServiceDistribution = Normal
    meanAcquireInterservice = 4.0
    sigmaAcquire = 0.75
    firingServiceDistribution = Normal
    meanFiringInterservice  = 1.0
    sigmaFiring = 0.2
     # in rounds per minute
    maxRateOfFire = 8
    thePER = 35
    thePED = 2
    numberRounds = 2520
    shooterRange = 30000
    burstRadius = 50
    numberGuns = 6
  Weights to determine the “best” shooter
    range = 0.125
    thePER = 0.125
    numberRounds = .5
    FieldArtillery = 0.25
    NSFS = 0.00

# Run execute properties
stopEvent = TargetArrival
stopEventCount = 425
```

Figure 2.8: IBCT COA 1A

(b) ***FCS***



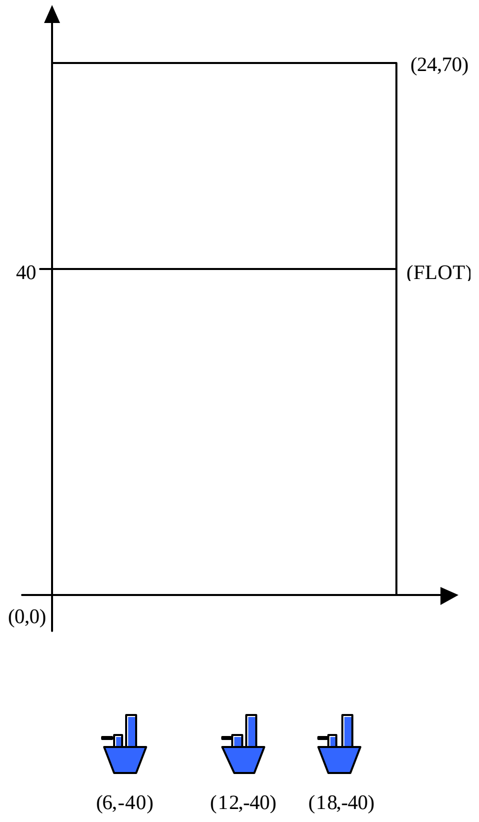
```
The AO properties
X distribution = Uniform(0,50)
Y distribution = Exponential(90)
Arrival distribution = Exponential(2.5)

Shooter properties
  meanProjectileVelocity = 100000 m/min
  Crusader Battery:
    acquireServiceDistribution = Normal
    meanAcquireInterservice = 2.0
    sigmaAcquire = 0.25
    firingServiceDistribution = Normal
    meanFiringInterservice  = 0.25
    sigmaFiring = 0.05
     # in rounds per minute
    maxRateOfFire = 12
    thePER = 1
    thePED = 1
    numberRounds = 2520
    shooterRange = 45000
    burstRadius = 50
    numberGuns = 4
  Weights to determine the “best” shooter
    range = 0.125
    thePER = 0.125
    numberRounds = .5
    FieldArtillery = 0.25
    NSFS = 0.00


# Run execute properties
stopEvent = TargetArrival
stopEventCount = 425
```

Figure 2.9: FCS COA 2A

2. NSFS Only

This COA serves as a base-line model for the effectiveness of indirect fires provided by naval assets. It involves running the simulation with the NSFS fire-power equivalent of one Army field artillery battalion in direct support of a maneuver brigade.

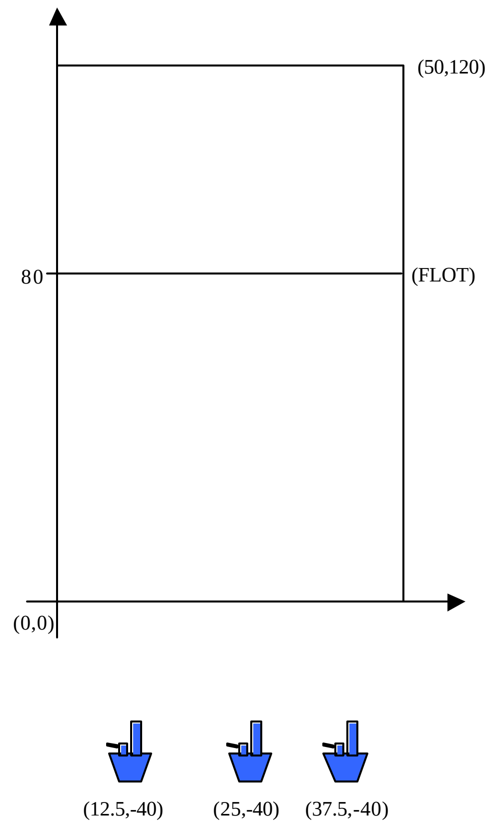
(a) IBCT



```
The AO properties
X distribution = Uniform(0,24)
Y distribution = Exponential(50)
Arrival distribution = Exponential(2.5)

Shooter properties
  meanProjectileVelocity = 70000 m/min
  DDG 51:
    acquireServiceDistribution = Normal
    meanAcquireInterservice = 5.0
    sigmaAcquire = 1.0
    firingServiceDistribution = Normal
    meanFiringInterservice  = 0.5
    sigmaFiring = 0.1
     # in rounds per minute
    maxRateOfFire = 24
    thePER = 100
    thePED = 2
    numberRounds = 2400
    shooterRange = 112000
    burstRadius = 75
    numberGuns = 2
  Weights to determine the “best” shooter
    range = 0.125
    thePER = 0.125
    numberRounds = .5
    FieldArtillery = 0.25
    NSFS = 0.00

# Run execute properties
stopEvent = TargetArrival
stopEventCount = 425
```

Figure 2.10: IBCT COA 1B

(b) ***FCS***



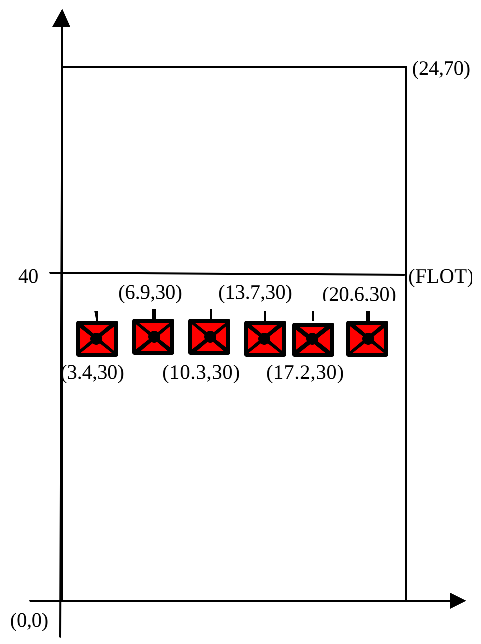
```
The AO properties
X distribution = Uniform(0,50)
Y distribution = Exponential(90)
Arrival distribution = Exponential(2.5)

Shooter properties
  meanProjectileVelocity = 100000 m/min
  DD21:
    acquireServiceDistribution = Normal
    meanAcquireInterservice = 2.5
    sigmaAcquire = 0.5
    firingServiceDistribution = Normal
    meanFiringInterservice  = 0.5
    sigmaFiring = 0.1
     # in rounds per minute
    maxRateOfFire = 24
    thePER = 10
    thePED = 2
    numberRounds = 2400
    shooterRange = 162000
    burstRadius = 50
    numberGuns = 2
  Weights to determine the "best" shooter
    range = 0.125
    thePER = 0.125
    numberRounds = .5
    FieldArtillery = 0.25
    NSFS = 0.00


# Run execute properties
stopEvent = TargetArrival
stopEventCount = 425
```

Figure 2.11: FCS COA 2B

3. Organic Army Fire Support with Reinforcing Army Fires

This COA involves adding one field artillery battalion to the base-line organic fire support model.

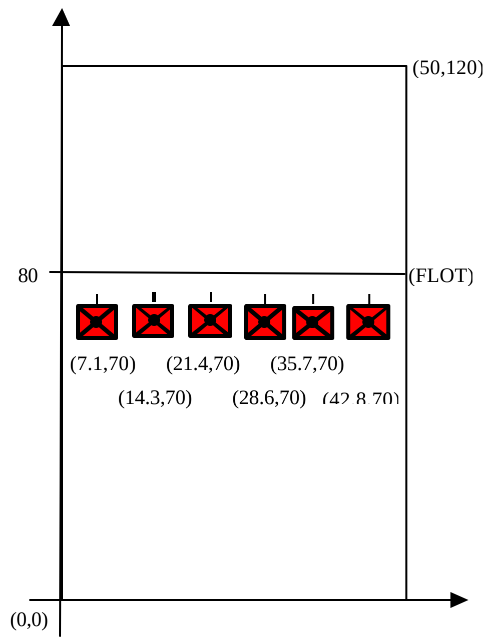
(a) IBCT



```
The AO properties
X distribution = Uniform(0,24)
Y distribution = Exponential(50)
Arrival distribution = Exponential(2.5)

Shooter properties
  meanProjectileVelocity = 70000 m/min
  Paladin Battery: See above
  Weights to determine the “best” shooter
    range = 0.125
    thePER = 0.125
    numberRounds = .5
    FieldArtillery = 0.25
    NSFS = 0.00

# Run execute properties
stopEvent = TargetArrival
stopEventCount = 425
```

Figure 2.12: IBCT COA 1C

(c) FCS



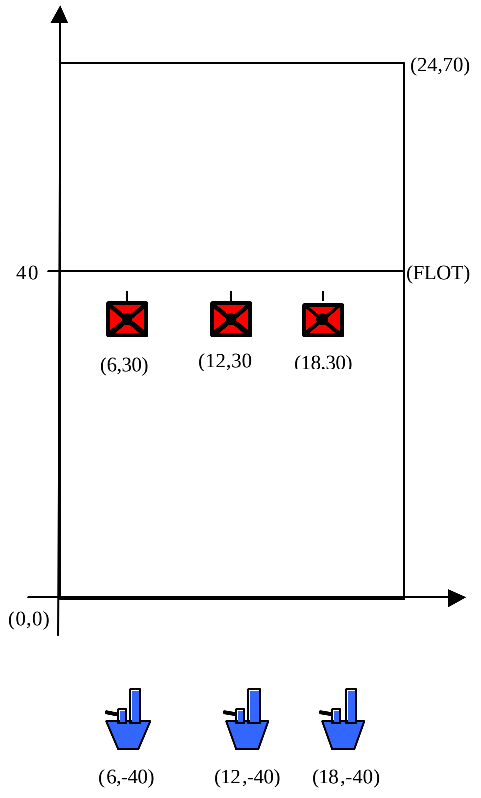
```
The AO properties
X distribution = Uniform(0,50)
Y distribution = Exponential(90)
Arrival distribution = Exponential(2.5)

Shooter properties
  meanProjectileVelocity = 100000 m/min
  Crusader Battery: See above
  Weights to determine the “best” shooter
    range = 0.125
    thePER = 0.125
    numberRounds = .5
    FieldArtillery = 0.25
    NSFS = 0.00


# Run execute properties
stopEvent = TargetArrival
stopEventCount = 425
```

Figure 2.13: FCS COA 2C

4. Organic Army Fire Support with Reinforcing Naval Fires

This COA involves adding the NSFS fire-power equivalent of one field artillery battalion to the base-line organic fire support model.

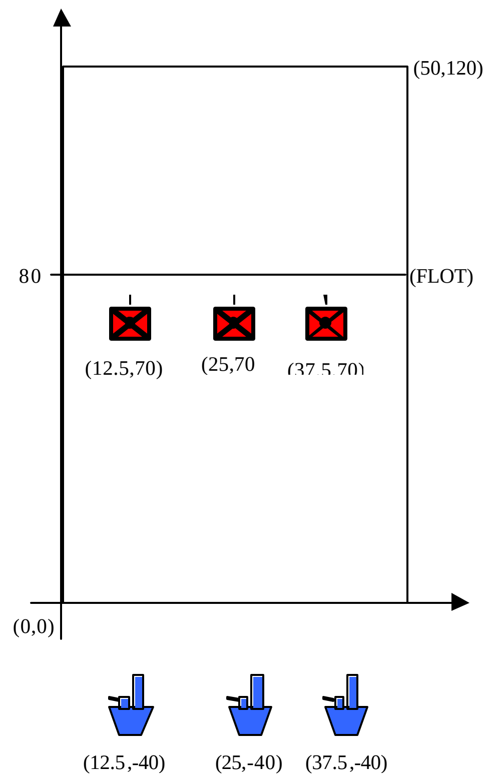
(a) ***IBCT***



```
The AO properties
X distribution = Uniform(0,24)
Y distribution = Exponential(50)
Arrival distribution = Exponential(2.5)

Shooter properties
  meanProjectileVelocity = 70000 m/min
  Paladin Battery:  See above
  DDG 51: See above
  Weights to determine the “best” shooter
    range = 0.125
    thePER = 0.125
    numberRounds = .5
    FieldArtillery = 0.25
    NSFS = 0.00

# Run execute properties
stopEvent = TargetArrival
stopEventCount = 425
```

Figure 2.14: IBCT COA 1D

(b) FCS



```
The AO properties
X distribution = Uniform(0,50)
Y distribution = Exponential(90)
Arrival distribution = Exponential(2.5)

Shooter properties
  meanProjectileVelocity = 100000 m/min
  Crusader Battery: See above
  DD21: See above
  Weights to determine the "best" shooter
    range = 0.125
    thePER = 0.125
    numberRounds = .5
    FieldArtillery = 0.25
    NSFS = 0.00


# Run execute properties
stopEvent = TargetArrival
stopEventCount = 425
```

Figure 2.15: FCS COA 2D

G. CONFIRMING THE ACCURACY OF THE MODEL

The accuracy of this simulation was confirmed entirely by the author. The process used was an iterative approach that included many ad-hoc techniques.

1. Input

(a) Weapons Characteristics

The data that were used for the individual weapons characteristics came from published sources on the individual weapons. Data that were not accessible, such as probable errors in range and acquisition times were "best guess" data that made sense. The overall data set used for each weapon is *not* 100% accurate, but it is fairly close to the actual data and is a suitable substitute to demonstrate the effectiveness of the FSST.

(b) Area of Operations and Battlefield

The area of operations consists of the size of the area and distribution of targets and target types. The parameters regarding the area of operations were estimated for each of the two scenarios from the current doctrinal battle space of an Army brigade. The estimate accounts for the role of the future IBCT and FCS brigade sized units.

The parameters for the distribution of targets are exponential through the depth and uniform across the breadth of the battle-space. The uniform distribution along the breadth of the battle-space models a brigade that fights along a front with the enemy attacking uniformly along that front. The exponential distribution models a higher likelihood of detecting targets towards the front of the battle-space, while accounting for enemy units that evade front-line defenses and are detected deeper within the unit's battle-space. The actual mean for the exponential distribution was derived by experimenting with several until the results made sense to the author.

Additional parameters for the model dictate how likely an enemy is to flee if not successfully engaged, the miss distance for a round to be considered errant, and the probability of an errant round causing collateral damage. All parameters used for the area of operations and the battlefield are shown below in Table 2.5.

Parameter	**IBCT**	**FCS**
Box: width x depth (km)	24 X 70	50 X 120
Distribution of Targets along width (front)	Uniform	Uniform
Distribution of Targets along depth	Exponential (mean = 50)	Exponential (mean = 80)
Arrival Intervals (min)	Exponential (mean = 2.5)	Exponential (mean = 2.5)
Flee Probability	0.9	0.9
Miss Distance for Errant Round	400	400
Probability of Collateral Damage by Errant Round	0.01	0.01

Table 2.5: Area of Operations and Battlefield Parameters

The parameters that determine the percentage of enemy units that are mechanized or light and the percentage of missions that are destroy, neutralize, and suppress were selected based on the experiences of the author. Although they do not represent a validated Army scenario, they provide the necessary information to demonstrate and test the FSST. They were depicted earlier in Tables 2.1 and 2.2.

(c) Attack Guidance

The attack guidance parameters dictate the number of munitions to fire at a specific target type. They were selected based on the author's experience and are shown below in Table 2.6 along with the parameters for a successful engagement.

(d) Success Parameters

These parameters determine the number of rounds that must score a hit on the target to inflict the damage necessary for a successful mission. A successful "hit" for a destroy or neutralize mission occurs when the round lands less than one burst radius away from the target. A successful "hit" for a suppression mission is when the round lands less than two burst radii away from the target. These parameters were selected based on the experience and intuitions of the author.

Target/Mission Type	**Engagement Criterion (rds)**		**Success Criterion (hits)**
	IBCT	**FCS**	
Armor			
Destroy	36	18	18
Neutralize	24	12	12
Suppress	3	3	1
Infantry in the Open			
Destroy	6	3	3
Neutralize	3	3	2
Suppress	3	3	1
Infantry Dug In			
Destroy	24	6	6
Neutralize	9	4	4
Suppress	3	3	1
Armored Personnel Carrier			
Destroy	24	12	12
Neutralize	18	9	9
Suppress	3	3	1
Light Skinned Vehicle			
Destroy	18	9	9
Neutralize	12	6	6
Suppress	3	3	1

Table 2.6: Target Engagement and Success Criterion

2. Output

The accuracy of the output from the model was checked in three separate ways throughout its development. They are discussed in detail in the following paragraphs.

(a) Common-Sense

At each stage in the development of the simulation the output was checked to ensure it made sense based on the input parameters. An example would be when determining the average time that it took for missions to be processed. A negative number would not make sense. That indicated a problem in the coding or the logic. Although not an entirely scientific approach, this method ensured that the answers made sense.

(b) Sequential

The sequential method pertains to checking a portion of the simulation as it is being developed using “canned” and oversimplified inputs. By checking each class against the desired output the simulation was verified in the most basic case.

As the simulation grew and more and more Java classes became interconnected, the actual answers to the output were no longer known. The experiences from repeatedly testing and running the simulation gave the author the insight needed to informally verify the output as the simulation became more and more complex. In this case the output was tested and verified in the same manner, except that instead of comparing the answer to a known solution it was simply checked to see if it made sense. For example, suppose the simulation consisted of 15 classes that seemed to run correctly. If another class or an enhancement of an existing class were being added that should refine the simulation, it would be run in the generic case where the refinement should make absolutely no difference. If that worked, the parameters would be adjusted to see if the enhancement did what it was intended to do. If it did, it was considered to function correctly.

(c) Parameterization

Varying the inputs to the model should make certain things occur. For instance, by increasing the range of a weapon system, the number of rejected missions should decrease. This type of verification was done for virtually all of the parameters. It is worth noting that in some cases what is expected and what actually occurs are not the same. This did not necessarily indicate that the simulation was not running correctly or capturing the effects of those parameters correctly. Instances where parameterized input did not yield outputs that made sense were investigated and corrected if needed. If it was determined that the particular instance actually was modeled correctly, the author generally found that the suspect output was the result of another parameter or modeling decision. For instance, as stated above increased range *should* result in less missions being rejected. If it did not, it *might* not mean the model is faulty. In one particular instance this modification resulted in a specific shooter running out of ammunition very early and not being able to cover his area of the battle-space. This resulted in an increase in the number of missions rejected, which does make sense. These anomalies provided insight that was unknown to the author earlier.

THIS PAGE INTENTIONALLY LEFT BLANK

III. ANALYSIS OF SIMULATION OUTPUT

A. OVERVIEW

Before analyzing the output, a review of the problem is in order. For the purposes of this paper, NSFS is artillery provided by naval platforms. Available data on naval and Army artillery indicate the strengths of naval platforms with respect to Army platforms are in extended range and rapid rates of fire. Inherent weaknesses are in the areas of probable error in range and deflection, time for coordination of fires, and time of flight due to stationing of naval platforms at least 40 kilometers from shore. Based on this information, the objective of this thesis is to determine the effectiveness of NSFS to the Army at brigade level and below in a littoral campaign. Stated another way, do the strengths of naval platforms outweigh the inherent weaknesses of those platforms for the purposes of providing fire support for Army units at brigade level?

The focus of the analysis of the data was to determine the effectiveness of NSFS to the Army brigade commander. Multi-attribute utility theory (MAUT) was used to determine the utility of a particular COA within each scenario. Using the MAUT procedure, the output from the simulation can determine the utility of each COA relative to the other COA's in that particular scenario. Effectiveness of NSFS can be measured by comparing COA's with and without NSFS.

By replicating each course of action, the simulation can produce confidence intervals on the utility of each COA. These data are useful because they give the user an idea of the robustness of a specific COA and allow the analyst to determine whether there is a significant difference in the utility between competing COA's. Fifty replications were made for each COA in order to get a range of output values. This range of output values was needed to compare the COA's using the analysis of variance techniques discussed later.

B. DESIGN OF THE EXPERIMENT

This experiment was designed to measure the effectiveness of NSFS using a common sense approach. The objective was to create and execute a design that was simple and could be easily understood by the end user, a military officer unschooled in operations research techniques. Because of this restriction, the design consisted of a relatively simple simulation program written in Java that provided easily interpreted output. The intent of the model was not to simulate every aspect of the indirect fire fight, but to simulate the more important aspects that are needed to determine the effectiveness of NSFS. The MOE's from the previous chapter determined the output needed for the simulation and the level of complexity with which to simulate the indirect fire battle.

1. Experiment

The experiment consisted of four different COA's or levels for each scenario, each with seven MOE's. Each of these COA's was replicated 50 times, for a total of 400 simulation runs producing a total of 2800 individual pieces of data. The data were collated and processed using the MAUT methodology weighting each MOE equally. This produced 50 sets of finalized output in the form of utility for each COA, or 200 separate bits of numerical output each of which was linked to a specific COA for each scenario. This data served as the baseline case to measure the utility of each COA. The experiment is depicted below in Table 3.1.

COA	**Replications**	
	IBCT	**FCS**
Army Only	50	50
Navy Only	50	50
Army w/ Reinforcing Army	50	50
Army w/ Reinforcing Navy	50	50
TOTAL	200	200

Table 3.1: Experimental Design

2. Data Analysis

Once the experiments were complete, a cursory look at the data was done for two reasons – to see if the output made sense (verification) and to see if any broad insight could be gained from it.

(a) Verification

The mean values for each of the MOE's for each COA in each Scenario are shown below in Table 3.2.

FMT	NAS	PF	TPS	PS	CD	CA	COA
IBCT							
11.1487	1.3174	0.7528	0.7071	0.9412	0.0000	0.8462	A
75.6035	0.4644	0.9956	0.3156	0.3170	3.2400	1.0000	B
7.3585	4.3069	0.7535	0.7098	0.9427	0.0000	0.8534	C
10.3692	3.5583	0.9976	0.7473	0.7491	1.7800	1.0000	D
FCS							
3.28651	2.12820	0.78983	0.54991	0.69632	0.0000	0.7414	A
10.1904	1.22583	0.9976	0.19579	0.19626	3.6400	1.0000	B
2.95028	5.11194	0.8024	0.54684	0.68154	0.0000	0.7478	C
3.62385	4.77277	0.97967	0.58253	0.59463	1.0200	1.0000	D

Table 3.2: Mean MOE Performance

COA A consists of three batteries of Army artillery, COA B consists of three ships, COA C consists of six batteries, and COA D consists of three batteries and three ships. As expected, the average fire mission times (FMT) declined and the number of available shooters (NAS) increased as more assets were added to the base case. Due to the extended range of Naval gunfire, it was expected that the area of the battlefield covered

(CA) and the percentage of missions fired (PF) for COA's with ships would be higher, and they are. Additionally, due to a combination of the lower accuracy of Naval guns and the extended ranges that Naval missions incurred, the number of mission causing collateral damage (CD) should have been higher for COA's with ships as well, and it was. Predictions for the success rate for missions fired (PS) and the total success rate for all arriving targets (TPS) are difficult to predict, but the expectation is that adding Naval gunfire to the equation would increase TPS due to the ability to fire targets, but PS would decrease due to the inaccuracy of the added firepower. Overall the data seems to indicate that the simulation was functioning correctly and yielding data that made sense.

(b) Insight

(1) Average Fire Mission Times. Due to the extended range of the missions fired by Naval gunfire, the fire mission times were severely degraded for COA's with NSFS. By adjusting the parameters for which asset to chose for firing missions, however, this can be mitigated. The objective is to improve the overall effectiveness of indirect fires. By adjusting the weighting of the variables that decide who fires what mission – range, PER, number of available rounds, and whether the asset is field artillery or NSFS, the simulation can use NSFS for those missions that it is most effective at engaging.

(2) Collateral Damage. COA's that included NSFS had a significantly higher incidence of collateral damage. Although not too surprising, it is important to keep in mind due to the sensitive nature of the the missions the US has been engaged in lately. To mitigate this, it is necessary to ensure that NSFS is used for missions that have low probability of such occurrences. This is possible on a tactical

level, but may be impractical when planning large-scale operations that synchronized all available indirect fires.

(3) Coverage Area. The coverage areas of COA's with NSFS was significantly higher than those without. The significance of this event is that it shows that NSFS can cover areas that traditional Army artillery cannot. In the scenarios created for this thesis, those areas are the rear and deep areas of the operation. Since Army artillery is traditionally pushed forward to support engaged units, units are not positioned to support rear echelon missions. Those missions are generally fired as targets of opportunity when they are in range. NSFS can cover that dead space, contributing to the overall success of the mission. Due to the limited range of Army artillery, it can generally not engage the enemy deep in the battlefield. NSFS can be used to engage targets before they enter Army artillery range to help shape the future battle for enaged units. These shaping missions can disrupt the enemy's assault, canalize him into prepared kill zones, and attrite him before he enters the close fight.

3. Analysis of Variance (ANOVA)

At first glance it would appear that a two-factor ANOVA test should be used to analyze the effects of COA and Scenario on the output. A closer look at the experiment however, shows that there are significant differences between like COA's in different scenarios. The locations of the assets, the size of the area of operations, and the specifications of the assets all differ immensely between scenarios. For these reasons, the author opted to perform two separate one-factor ANOVA tests to test the null hypothesis (H_o) that the treatment (COA) means were identical, or that total utility is not affected by COA selection (Devore, 390-391). For both scenarios, the null hypothesis is rejected, indicating that there was a statistically significant difference between at least one of the treatments and the remainder of the treatments in each scenario.

To determine which, if any, of the COA's were different from one another, multiple comparisons was used. The question that remained to be answered was to decide for each i and j whether $\mu_i = \mu_j$. Tukey's procedure, the T Method, was used to produce a collection of simultaneous confidence intervals about the true value of all differences μ_i and μ_j. If the confidence interval for a specific difference contained the value zero, then those two samples were deemed equal at level α of the experiment. For this experiment, α was set at 0.05 (Devore, 401-402).

This analysis was performed using SPLUS and indicated that the differences between all treatments were significant. For both scenarios the COA's containing Army field artillery with reinforcing Naval gunfire were superior. The output for the T Method is shown below in Tables 3.3 and 3.4.

	Estimate	Std.Error	Lower Bound	Upper Bound	Includes Zero
1A-1B	0.846	0.0393	0.744	0.948	NO
1A-1C	-0.799	0.0393	-0.901	-0.697	NO
1A-1D	-1.970	0.0393	-2.070	-1.860	NO
1B-1C	-1.640	0.0393	-1.750	-1.540	NO
1B-1D	-2.810	0.0393	-2.910	-2.710	NO
1C-1D	-1.170	0.0393	-1.270	-1.060	NO

Table 3.3: IBCT COA Comparison using T Method

	Estimate	Std.Error	Lower Bound	Upper Bound	Includes Zero
2A-2B	1.090	0.042	0.983	1.200	NO
2A-2C	-0.782	0.042	-0.891	-0.673	NO
2A-2D	-2.080	0.042	-2.190	-1.970	NO
2B-2C	-1.870	0.042	-1.980	-1.770	NO
2B-2D	-3.170	0.042	-3.280	-3.060	NO
2C-2D	-1.300	0.042	-1.410	-1.190	NO

Table 3.4: FCS COA Comparison using T Method

The boxplots of the utility for the different COA's are shown below in Figures 3.1 and 3.2. They show the performance of each COA by depicting the median as a stripe. The box contains both the upper and lower quartiles. The whiskers of the boxplot are depicted by the brackets which contain the lower and upper bounds of the data defined as 1.5*(Inter-Quartile range). Outliers are shown as stripes outside of the bracketed data. They are a useful graphical depiction of the performance of the different COA's that give the user an easy to interpret depiction of the data.

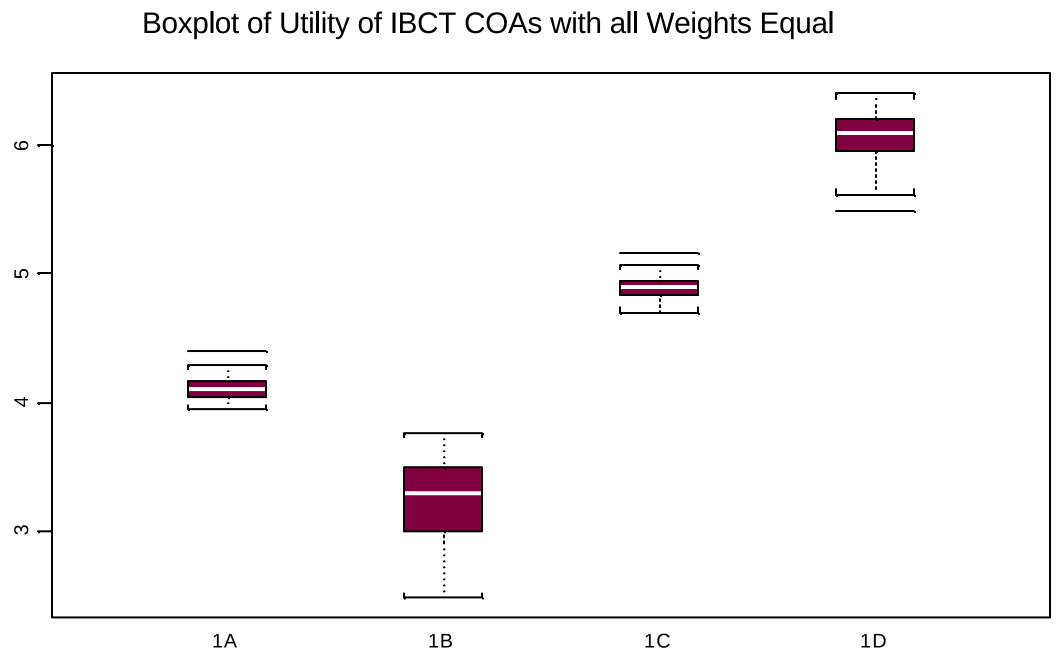


Figure 3.1: Boxplot of IBCT COA's

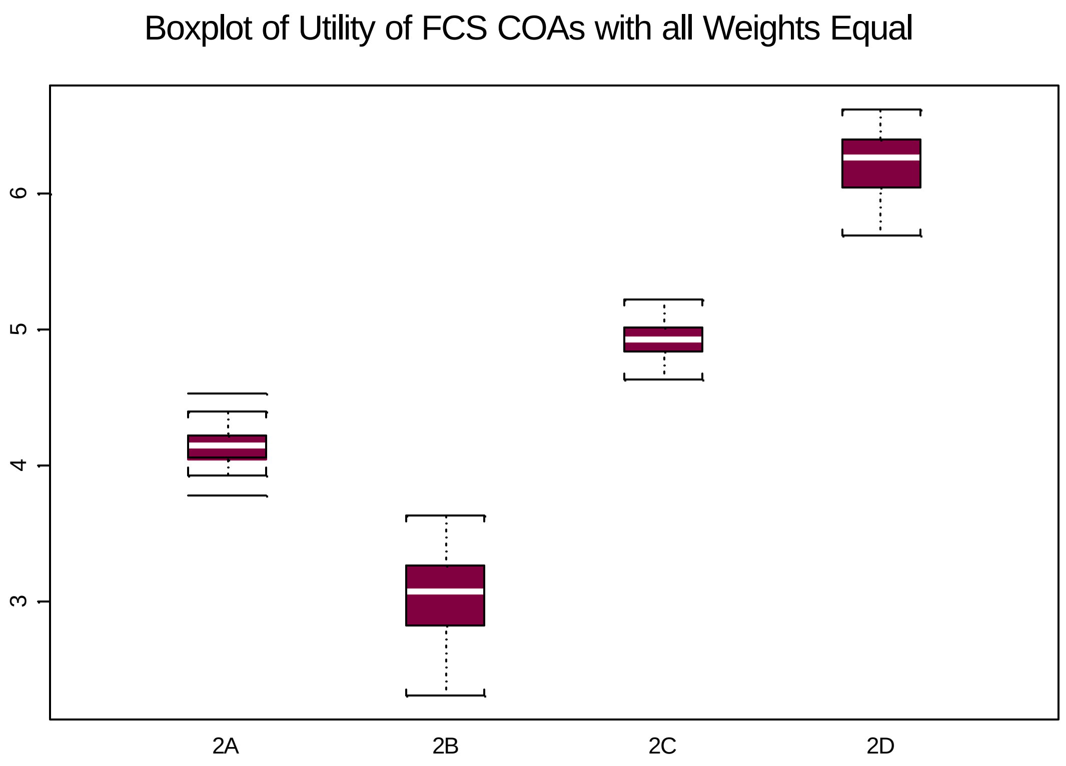


Figure 3.2: Boxplot of FCS COA's

The final output (the T Method comparison tables and the boxplots) from the experiment meets the requirements of simplicity. Most people understand the theory behind confidence intervals and can comprehend the basic idea behind Tukey's comparisons of μ_i and μ_j. The boxplots, although somewhat simplistic, provide a simple, graphical depiction of each COA's median utility and range.

4. Sensitivity to Weights

For this particular experiment, all MOE's were weighted equally. The actual weighting of each MOE is best left to the professional judgement of the leaders and staffs involved in the conflict. Equal weighting results in each MOE having about 14% of the total weight. To demonstrate the sensitivity that the analysis has to the weighting of each of the MOE's, one MOE was given 50% of the total weight, resulting in each remaining MOE having roughly 8.33% of the remaining weight. This procedure was replicated seven times (once per MOE). The boxplots for each of those branches from the base case is shown below in Figures 3.3 and 3.4.

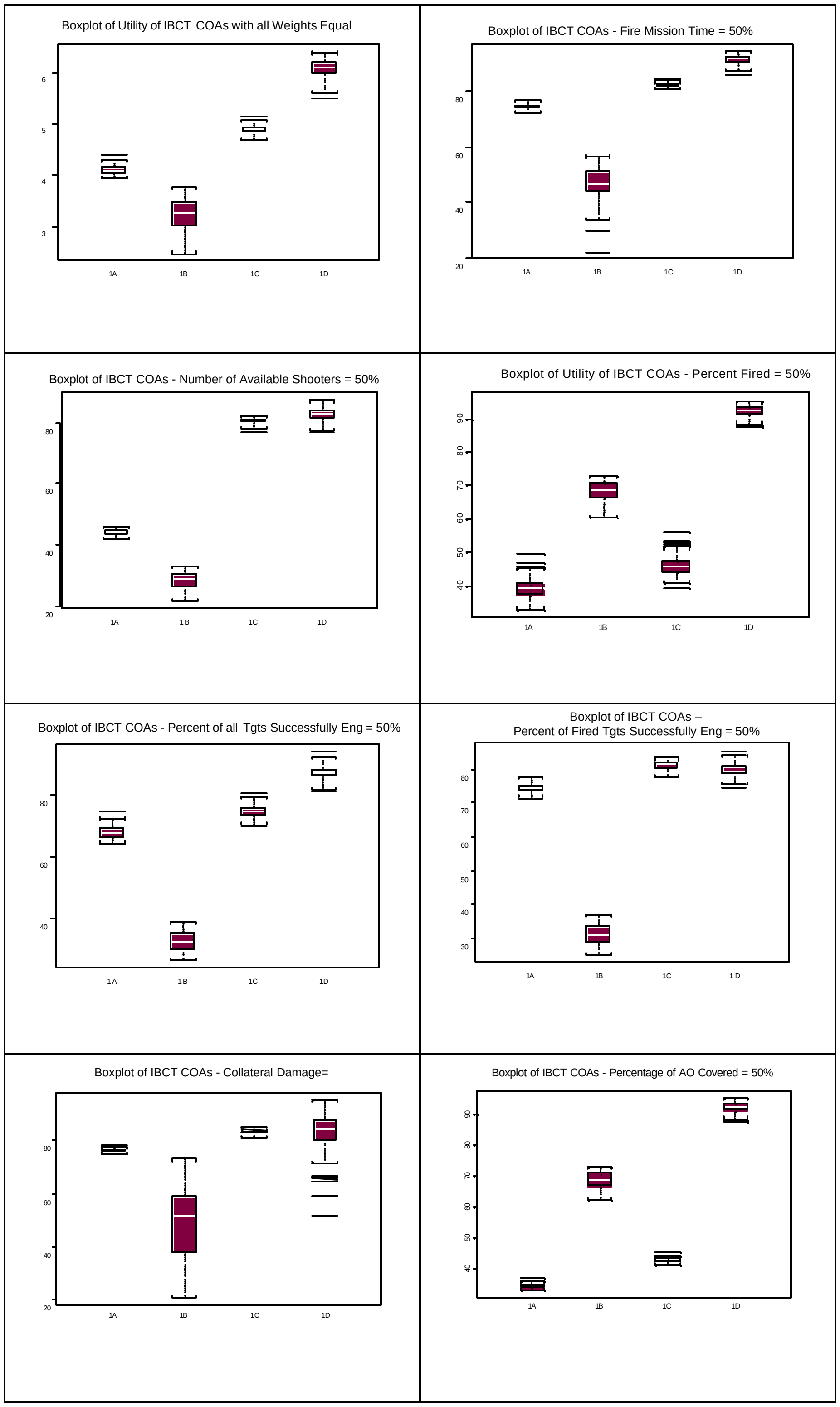


Figure 3.3: Boxplots of Different Weightings for IBCT

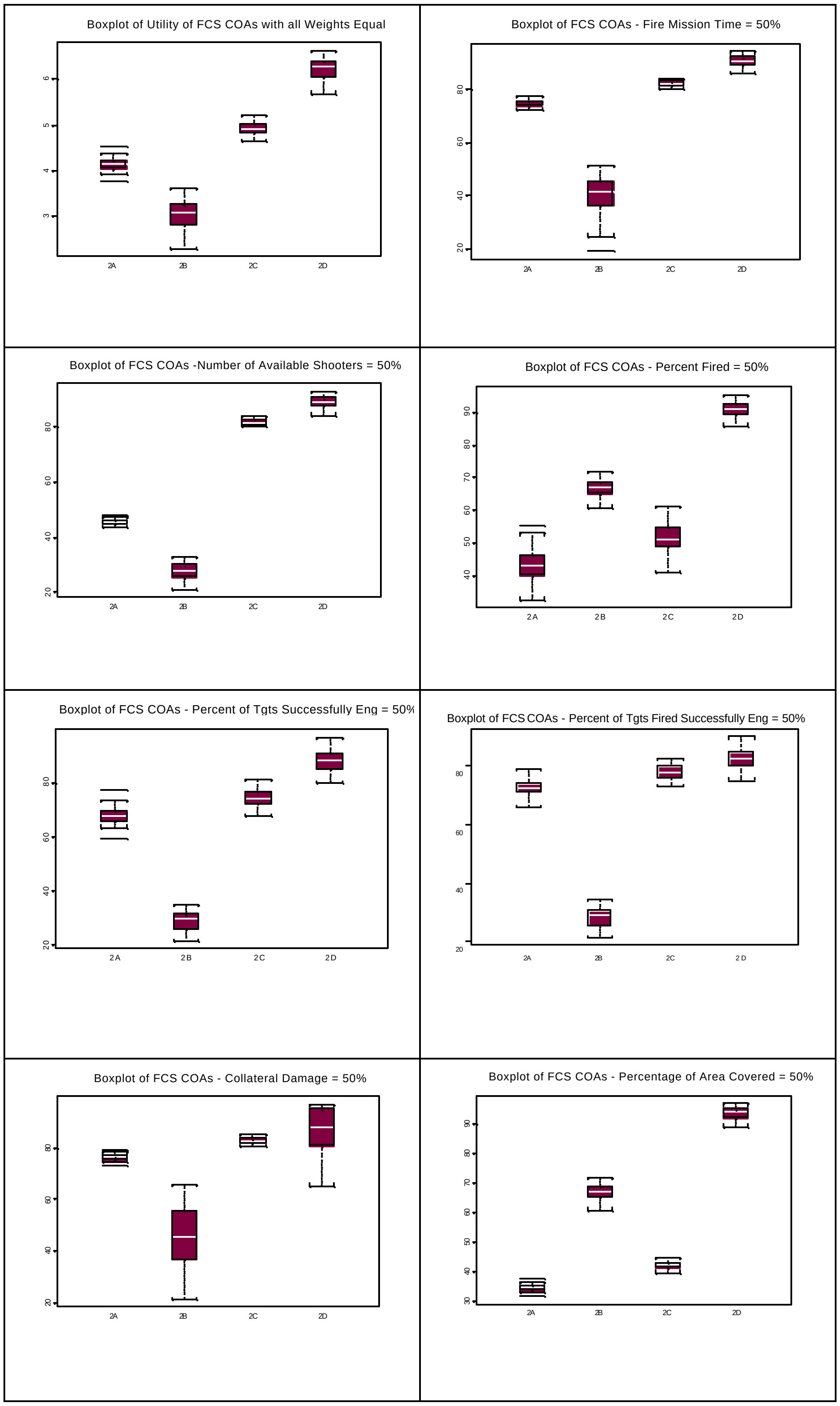


Figure 3.4: Boxplots of Different Weightings for FCS

These boxplots show how the utility for the COA’s in the IBCT and FCS scenarios are affected by changes in the weights of the various MOE’s. This output is helpful because it gives the user an idea of the sensitivity that the output has to the utility. In addition to the medians of the utility shifting up and down, changing the weights affects the spread too.

Looking at a specific boxplot – IBCT with Collateral Damage weighted at 50% we notice that the COA 1C and 1D (organic Army fires with reinforcing Army and Navy fires) both appear to have similar means. In fact, the T Method confirms that there is no statistical difference in the means for those two treatments as shown below in Table 3.5. In this specific COA using the weights listed above, the commander has more to consider than he might if the utility of the COA’s were significantly different, as in the base case. Although both COA’s are similar, the variances of the overall utility of each COA are much different and could be an important factor in his decision.

	Estimate	Std.Error	Lower Bound	Upper Bound	Includes Zero
1C-1D	0.881	1.64	-3.36	5.12	YES

Table 3.5: Comparison of IBCT COA 1C and 1D (Collateral Damage = 50%)

Changes in the weighting scheme for a single MOE result in linear changes to the utility. However, the effects of higher order changes (changing the weights of two or more MOE’s simultaneously) have not been fully explored. It is suspected that higher order changes will result in monotonistic changes in the overall utility (e.g. if changing the weight of MOE’s one and two individually shift the utility in favor of a particular COA, then increasing the weight of both simultaneously *should* shift the utility in favor of the same COA). Shown below is a boxplot of the effect of changing the weights of collateral damage and fire mission times, both of which favor COA 1D, to 80% of the total utility (40% each) for the IBCT scenario. The remaining MOE’s are now worth a mere 4% of the total utility. The boxplot of the results shown below in Figure 3.5 now seems to indicate that COA 1D is the preferred COA. The T Method results shown below in Table 3.6 indicate that there is no significant difference between COA's 1A and 1D using this weighting scheme. However, the COA with the most utility is COA 1C, which is evident by the fact that the confidence interval for 1A – 1C is negative, and 1C – 1D is positive.

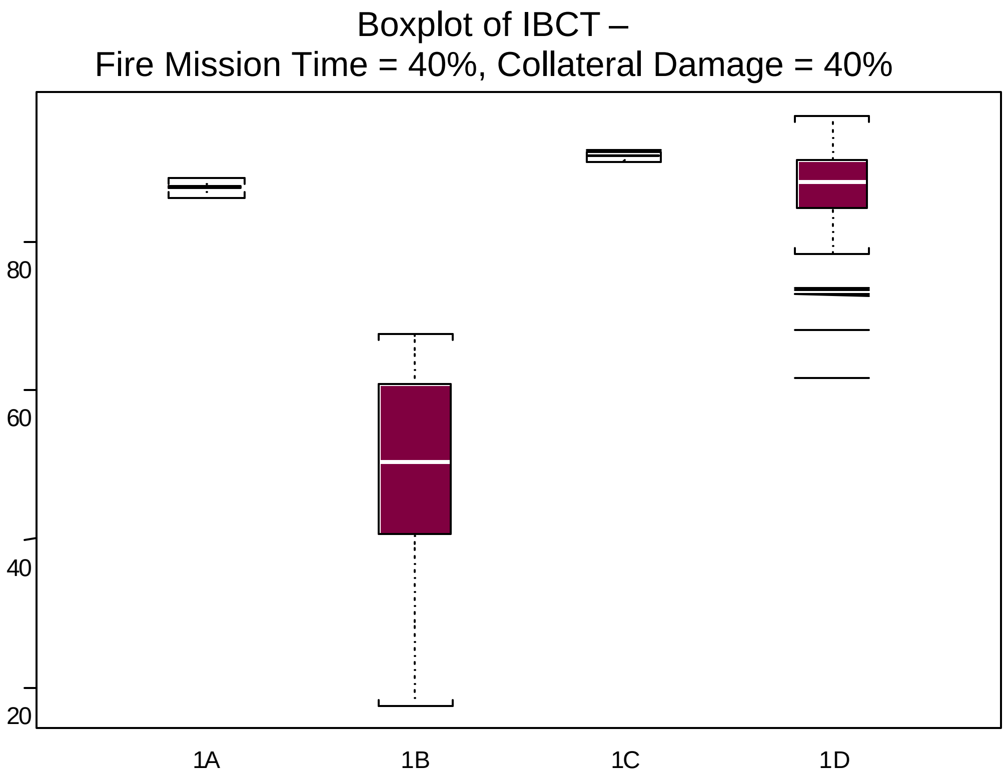


Figure 3.5: Boxplot of IBCT Utility

	Estimate	Std.Error	Lower Bound	Upper Bound	Includes Zero
1A-1B	37.70	1.42	33.90	41.400	NO
1A-1C	-4.23	1.42	-8.02	-0.448	NO
1A-1D	1.08	1.42	-2.70	4.860	YES
1B-1C	-41.90	1.42	-45.70	-38.100	NO
1B-1D	-36.60	1.42	-40.40	-32.800	NO
1C-1D	5.31	1.42	1.53	9.100	NO

Table 3.6: Comparison of IBCT COA's (Fire Mission Time = 40%, Collateral Damage = 40%)

This analysis does not produce specific answers as to what the effectiveness of NSFS is. Rather it shows that the answer is a function of the inputs to the COA's and scenarios and the weights attributed to each MOE. If the commander weighted all MOE's equally and relied entirely on the output from this experiment, the logical choice for both the IBCT and FCS scenarios would be to use the organic Army Artillery Battalion in direct support reinforced by Naval gunfire. Different scenarios will certainly result in different results.

C. LIMITATIONS OF THE ANALYSIS

1. Input and Output

This analysis is built upon a number of significant assumptions, the first of which is that the simulation accurately models and measures the MOE's. Additionally, as with any simulation or analysis, the input variables dictate the output. Several assumptions and "best guesses" were made to get reasonable input variables. Some of the input variables, such as number of rounds required to destroy, neutralize, or suppress enemy armor are classified information. Others, such as probable errors in range and deflection for weapon systems were estimated for several reasons. First, since they vary with range and propellant type and amount used, they are average measures. And secondly, since the actual tables with the actual data are not available to the general public, best guess data was used.

2. Decision Aid

The FSST and the analysis presented here are meant as a decision aid. They are not intended to replace decision-makers or leaders. Those individuals are charged with the ultimate responsibility for what occurs as a result of their decisions. Their experience coupled with the analysis presented here will help guide them to make the best decision with respect to the circumstances.

Due to the nature of this model, it does not measure the intangible aspects associated with the effectiveness of NSFS. Although we purport to measure reliability, flexibility, and lethality, those measurements are a function of the definitions presented in this thesis and cannot replace reality. They simply provide insight into what measurements of those topics might actually be.

IV. CONCLUSIONS

A. RESULTS

The results of this thesis indicate that NSFS can be effective in providing support for Army units at brigade level in a littoral campaign for the IBCT and the FCS scenarios. The measures of effectiveness used for this analysis were fire mission times, available firing platforms, percentage of missions fired, percentage of successful missions based on all arriving targets, percentage of successful missions based on missions fired, number of rounds that caused collateral damage, and the percentage of the area of operations covered. COA's were developed consisting of different artillery task organizations of Army and Navy artillery for each scenario. Using the Fire Support Simulation Tool developed for this thesis, each COA was simulated 50 times to get values for the MOE's.

Using multi-attribute utility theory each MOE for each COA replication within a particular scenario was given a utility rating based on the actual value of that MOE/COA combination compared with other MOE/COA combinations in the scenario. The final utility for a specific COA replication in a scenario was determined by weighting and then combining the utility of the MOE's for each COA replication in a particular scenario. Single factor ANOVA and Tukey's procedure for multiple comparisons were used to determine whether there was a statistically significant difference among the COA's.

When using equal weights for all of the MOE's, the best COA for both scenarios was the COA that consisted of a mixture of Army and Navy artillery. Based entirely on the weapon systems specifications used, this result indicates that there could be a scenario for the IBCT and FCS where NSFS adds more utility and is effective as a fire support weapon in a littoral campaign.

B. ISSUES

This analysis revealed that there is merit to the use of NSFS in place of Army artillery for reinforcing fires in the littorals. However, nothing can truly replace the feeling of ownership that a unit commander feels by having supporting elements on the ground with him and within arm's reach. Most if not all leaders feel very comfortable with the capabilities and limitations of those assets they are most familiar with. Joint training exercises between Naval and Army units are extremely rare. Lack of that joint training and personal prejudices build distrust, which may prove an insurmountable obstacle if not corrected.

Nothing speaks more highly of dedication to success of a mission than having a personal stake in the outcome of the mission. Most soldiers stake their lives on the successful completion of their missions. If assets providing support in the form of naval gunfire do not have that same stake, Army commanders feel uncomfortable.

The results of this analysis assume the relatively efficient use of available assets to accomplish all fire missions. If assets such as naval gunfire can be called away at a moment's notice, their reliability to the commander on the ground becomes suspect. This leads to inefficient use of the asset. Basically, the Army commander will try to get as much as he can out of that asset before it gets taken away. This type of misuse will almost certainly create prejudice and mistrust among the Navy towards the Army.

These issues must be addressed before naval gunfire can be integrated seamlessly into Army operations now or in the future. Digital synchronization, future technologies, and memorandums of agreement will help, but alone they will not suffice. Soldiers know that the soldier across the street will be ready because they see him and her working and training every day. These issues of mistrust must be worked out on the ground, commander to commander, sergeant to petty officer, and soldier to sailor.

C. RECOMMENDATIONS FOR FURTHER RESEARCH

Research on the effectiveness of NSFS to Army commanders has provided one perspective and one answer to an extremely complex and ever-changing problem. As with much significant research, more questions than answers resulted, which is the basis for the following recommendations for further study of this problem.

1. Further Development of the FSST

For the purposes of this thesis the FSST was adequate. Further development could provide users with a higher fidelity and a more versatile tool. The current version of the FSST does not have a graphical user interface (GUI) and is limited in the distributions that can be used to determine the locations of arriving enemy targets. It also does not account for different times of flight for different weapon systems and different ranges, and accounts for only one type of ammunition. These are just a few areas that could use improvement, and will be discussed in further detail below

(a) Graphical User Interface and Target Distribution Model

(1) GUI. A GUI could be developed to simplify the development of scenarios. The GUI could prompt the user for all needed information and could have built in error checking mechanisms to prevent the user from entering bad data. This type of enhancement would make the Java underpinnings of the simulation transparent to the user and would make the tool more likely to be used for experimentation and analysis.

(2) Develop Unique Distributions to Be Used With the GUI

- Bivariate Triangular. The current version of the FSST uses two parameter distributions for the breadth of the battlefield and one parameter distributions for the depth of the battlefield. This is due to the fact that the FSST was designed for a fairly specific purpose. One of the strengths of Java is that it is allows for easily extending the capabilities and flexibility of any program. Making the FSST generic enough to accept a broader set of possible distributions for the depth and breadth of the battlefield would be relatively simple and would give the user a much more powerful tool. To allow for the analysis of future scenarios that might include a strongpoint-type position deep in enemy territory, the development of distributions that make the probability of encountering enemy targets at edges of the FLOT higher than in the center would be appropriate (see Figure 4.1).

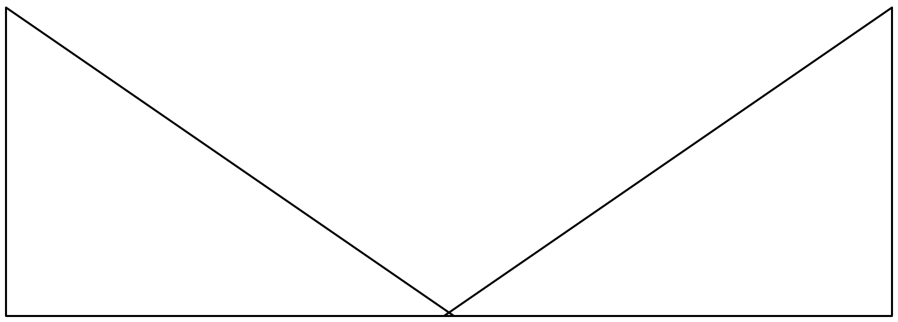


Figure 4.1: Unique Dual Triangular Distribution

Using this distribution bivariately would create a distribution that could model a strongpoint-type defense. The objective is to create a distribution that looks like an inverted, four-sided pyramid (Figure 4.2). The bivariate version of the dual triangular distribution would probably look more like a tarp suspended by all four corners and pulled taut in the center that sags somewhat in the center of each side.

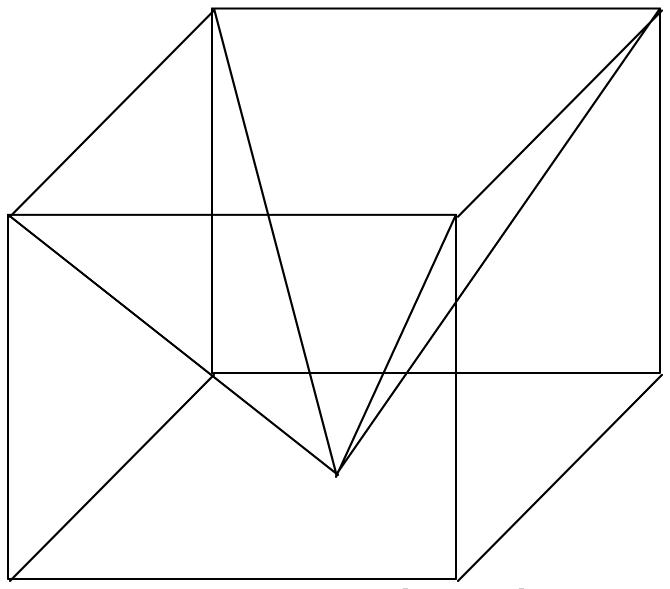


Figure 4.2: Inverted Pyramid

Shown below in Figure 4.3 is what an example of this type of bivariate distribution might look like, where the probability of encountering the enemy is highest on the perimeter and lowest at the center. Here we depict an Army unit in a desert-type environment near the coast surrounded by enemy units. The unit has organic fire support, but cannot effectively cover his perimeter with them. NSFS can provide the extra fire support needed to cover the entire area of operations.

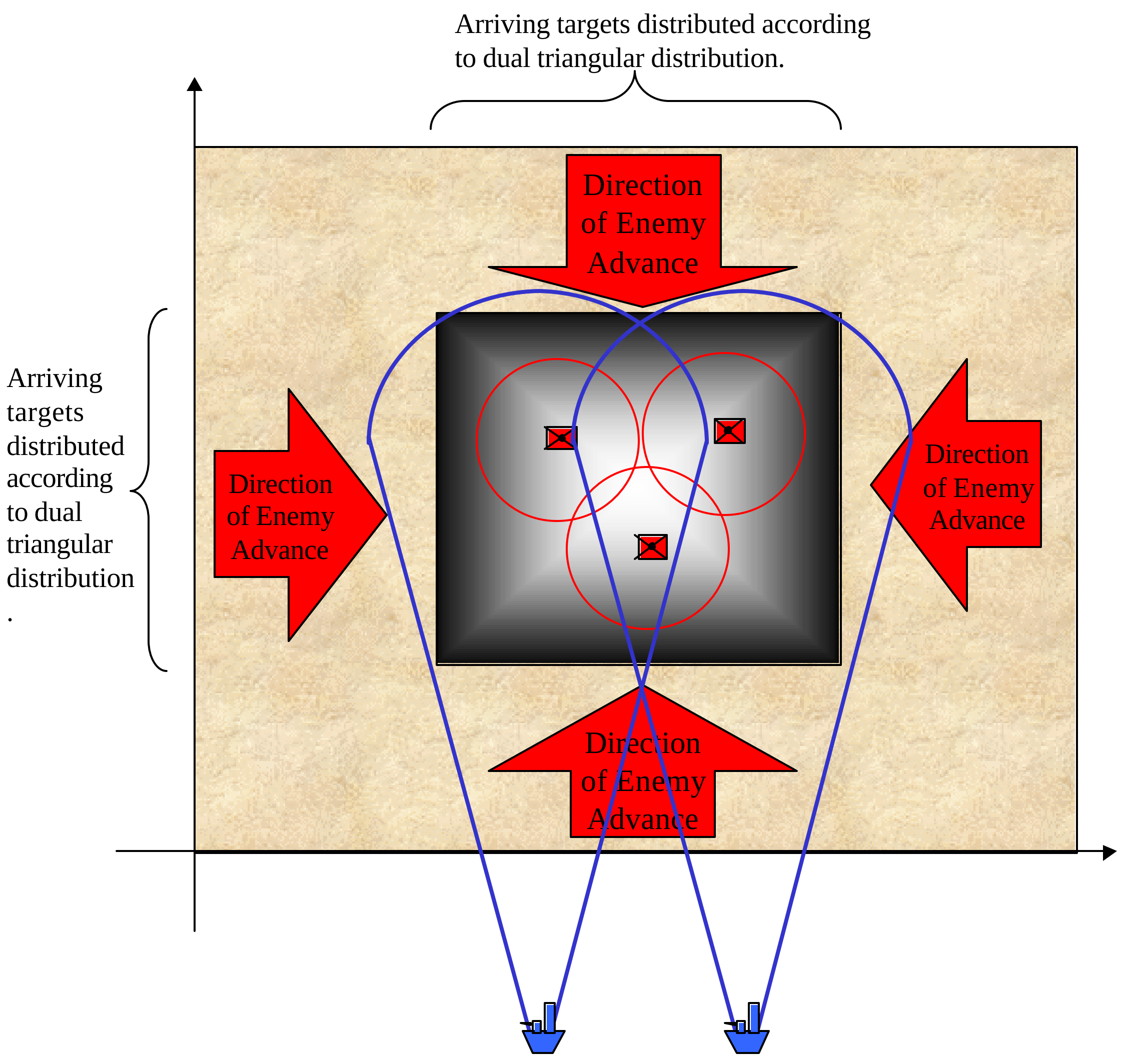

Arriving targets distributed according to dual triangular distribution.
Arriving targets distributed according to dual triangular distribution.
Direction of Enemy Advance
Direction of Enemy Advance
Direction of Enemy Advance
Direction of Enemy Advance

Figure 4.3: Picture of Density of Targets on the Battlefield. Dark indicates higher density

This distribution does not have to be symmetrical. It is conceivable that a unit could be surrounded by enemy units with different characteristics on each side. In this case, these enemy forces might attack the surrounded unit at different rates on each side. By adjusting the parameters of the bivariate dual triangular distribution, the user could model aggressive units to the left and front attacking in conjunction with less aggressive units to the right and rear. An example of what this might look like is shown below in Figure 4.4.

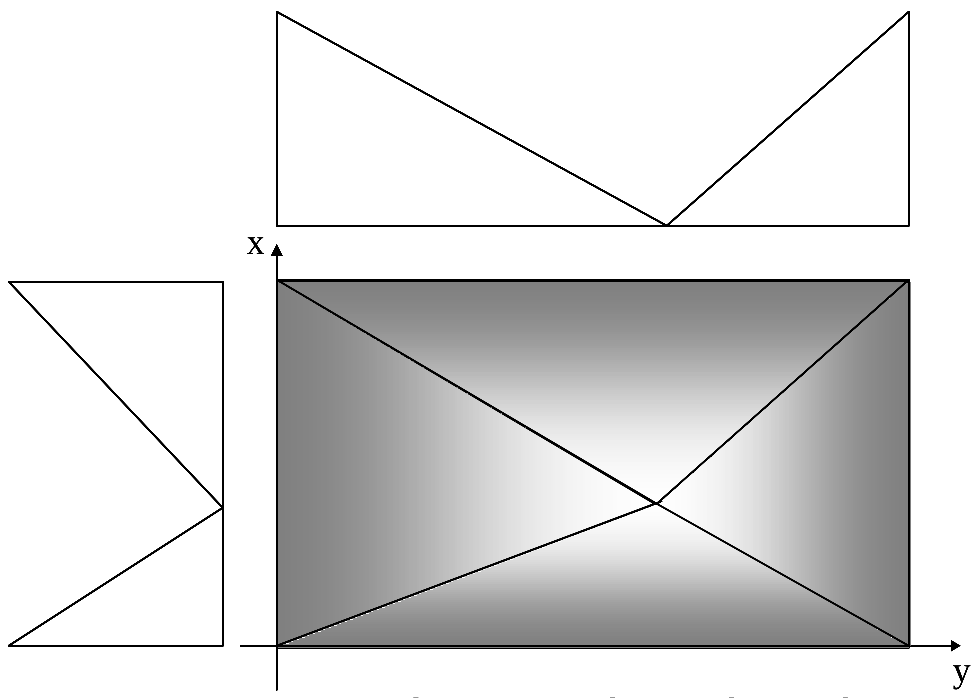


Figure 4.4: Un-symmetrical Bivariate Dual Triangular Distribution

By incorporating unique distributions such as this and others, we can more realistically model a broader array of scenarios. These distributions should be tied into the GUI described earlier to give the user a simple method of picking distributions and their parameters.

- Polar Coordinates. Another option for developing a model that depicts a strongpoint defense would be to model the arrival of targets using polar coordinates. In this case the two variables of angle and radius could generate a bivariate distribution that looks like a bowl or a ripple in a pond (see Figure 4.5, radially fleeing target). The randomly chosen polar coordionates that follow the particular bivariate distribution could then be converted to Cartesian coordinates for the simulation. An example of a very simple distribution would be one in which the angle was uniform between 0 and 6.28 radians, and the radius or range were chosen based on a right triangular distribution. This distribution would look like an inverted cone.

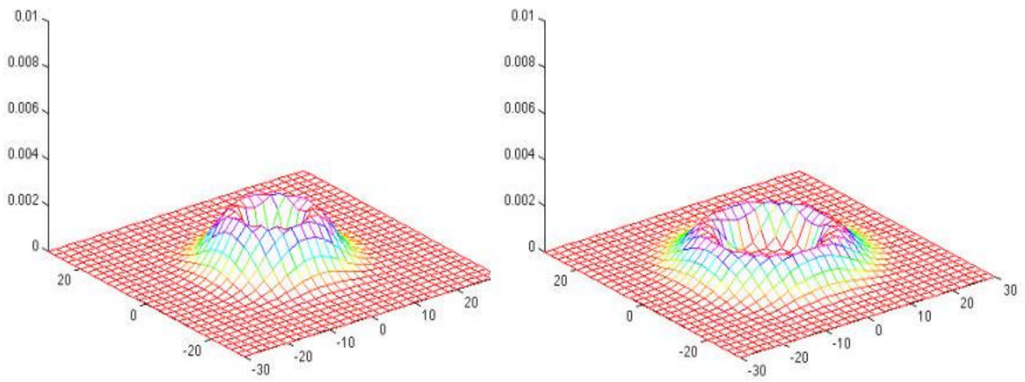


Figure 4.5: Sample polar coordinate bivariate distributions (From: Eagle).

(b) Survivability

The current version of the FSST discounts the effects of survivability, as fire support systems are not attrited in the model. By accurately accounting for attrition from counter-battery fire, mines, enemy actions, and mechanical failure, the FSST could be an improved model of the battlefield. This enhancement would enable the user to measure, compare, and analyze the effects of more survivable systems using a relatively simple tool. A possible thesis topic would be an analysis of the effectiveness of different howitzer systems for the IBCT and FCS-OF.

(c) Expand the Types of Ammunition the Simulation Models

Expanding the family of munitions that the simulation accounts for would make the FSST much more complex than it already is, and could make scenario development more time consuming (if the user wanted to use more than one type of ammunition). The benefits of this would be that it would allow comparisons between different basic loads of munitions to optimize success on the battlefield. This would also allow for more realistic comparisons among artillery systems by accounting for the availability of different munitions for different systems. Overall, this improvement would make the simulation more flexible and realistic.

(d) Modify Modeling of Time of Flight

Time of flight is calculated based on the average velocity of projectiles regardless of range or delivery system. To make the simulation more accurate, another method of calculating time of flight should be devised. One possible solution would be to take tabularized firing tables for Army and Navy artillery to get the times of flight and plot those times based on range for different weapon systems and use regression to develop a simple formula to compute times of flight for individual platforms.

2. Fire Support Optimization Based on Threat

Mission, enemy, time, terrain, and training make all military campaigns unique. Based on these factors, the force structure for a military campaign is determined by leaders who generally use intelligence estimates and personal experience as their guide. Using the FSST, a student could design an experiment to determine the optimal mix of Naval and Army artillery for a particular campaign. This analysis would provide the decision-maker a quantitative basis to aid in the decision-making process.

This problem can be examined using several different techniques. One technique would be to design a multi-factorial experiment and use FSST to determine the optimal strategy by using regression or ANOVA techniques to map response surfaces. Another technique would be to use linear and non-linear optimization techniques to determine the optimal solution. Certainly other techniques exist, yet these are two simple well-developed procedures that are available.

3. Cost Analysis of Army Fire Support Systems

Effectiveness of NSFS in this analysis is based entirely on utility to the Army brigade commander. The brigade commander is not concerned with cost because it does not affect him or his operation directly – generally, if the assets are available, they will be made available to him. However, when developing the Nation's future force, cost is an enormous factor and cannot be discounted.

An excursion from this thesis would be to optimize future fire support systems based on cost. By using a set of approved Army scenarios for the future, the student could design an experiment to determine the utility of different combinations of fire support systems throughout the Army and then optimize that force with respect to utility and cost. Put in more simplistic terms, this analysis would be a bang for the buck analysis of Army fire support.

Another possible excursion would be to analyze and optimize utility and cost for a specific scenario that took into account cost. For example, the fire support for a purely littoral Army operation could be optimized for utility given a limited budget. By experimenting with different feasible combinations of NSFS and Army artillery, the student could optimize the assets provided for that operation for a specific budget.

D. SUMMARY AND CONCLUSIONS

The effectiveness of Naval Surface Fire Support is difficult to define and even harder to measure. By using value systems design to define effectiveness of NSFS in a hierarchical manner using an objective tree, NSFS effectiveness can be defined quantitatively by the lowest sub-objectives on the objective tree. Those lowest level, quantitative sub-objectives are the MOE's that were used to measure the effectiveness of NSFS. Figure 4.6 shows the condensed objective tree that depicts the main objective, the first level objectives, and the lowest level objectives or MOE's (notice that all other intermediate level objectives have been removed).

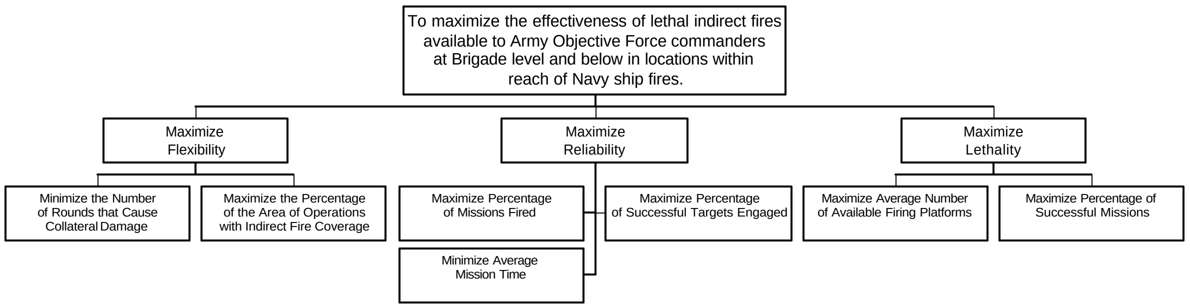

To maximize the effectiveness of lethal indirect fires available to Army Objective Force commanders at Brigade level and below in locations within reach of Navy ship fires.
Maximize Flexibility
Maximize Reliability
Maximize Lethality
Minimize the Number of Rounds that Cause Collateral Damage
Maximize the Percentage of the Area of Operations with Indirect Fire Coverage
Maximize Percentage of Missions Fired
Maximize Percentage of Successful Targets Engaged
Minimize Average Mission Time
Maximize Average Number of Available Firing Platforms
Maximize Percentage of Successful Missions

Figure 4.6: Condensed Objective Tree

By combining the strengths of Army artillery and naval gunfire, an Army Brigade commander can organize a fire support team that is better able to support his missions. The task organization of assets used is strongly dependent on the weighting scheme the commander adopts for the MOE's presented earlier. Although there is evidence supporting the use of NSFS in a support of Army operations in the littorals, there are a myriad of issues suc h as training, mistrust, and synchronization that must be addressed to make these types of joint campaigns successful. In the final analysis, it was determine that there is strong *quantitative* and *analytical* evidence to support the effectiveness of NSFS to an Army Brigade commander engaged in a littoral campaign.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX 1: IBCT RAW DATA

FMT	NAS	PF	TPS	PS	CD	CA	COA
10.1155	1.3809	0.7412	0.6991	0.9455	0	0.8462	1A
10.0987	1.3208	0.7529	0.7224	0.9594	0	0.8462	1A
13.5736	1.2861	0.7576	0.6981	0.9221	0	0.8462	1A
10.6466	1.2771	0.8047	0.7671	0.9532	0	0.8462	1A
8.7089	1.3835	0.7553	0.7059	0.9346	0	0.8462	1A
9.2294	1.3523	0.7553	0.7088	0.9429	0	0.8462	1A
8.8930	1.5523	0.7247	0.6746	0.9369	0	0.8462	1A
13.1867	1.2501	0.7647	0.7268	0.9533	0	0.8462	1A
10.1874	1.4308	0.7365	0.6888	0.9385	0	0.8462	1A
10.8690	1.2736	0.7576	0.7275	0.9624	0	0.8462	1A
8.7823	1.4471	0.7388	0.7028	0.9521	0	0.8462	1A
12.8423	1.2563	0.7882	0.7177	0.9146	0	0.8462	1A
11.2220	1.3108	0.7459	0.7014	0.9427	0	0.8462	1A
10.1027	1.2453	0.7529	0.7092	0.9434	0	0.8462	1A
12.7777	1.2948	0.7647	0.7187	0.9412	0	0.8462	1A
13.5144	1.2354	0.7529	0.7082	0.9406	0	0.8462	1A
10.8664	1.3880	0.7788	0.7352	0.9453	0	0.8462	1A
10.2689	1.4459	0.7459	0.6894	0.9243	0	0.8462	1A
10.3266	1.4044	0.7341	0.6934	0.9453	0	0.8462	1A
9.7868	1.4779	0.7412	0.6862	0.9243	0	0.8462	1A
11.0182	1.2850	0.7435	0.7038	0.9489	0	0.8462	1A
13.4077	1.2219	0.7694	0.7262	0.9472	0	0.8462	1A
12.7512	1.1467	0.7929	0.7458	0.9453	0	0.8462	1A
9.8451	1.3921	0.7553	0.7254	0.9596	0	0.8462	1A
9.5355	1.4259	0.7553	0.7092	0.9404	0	0.8462	1A
12.0433	1.2273	0.7459	0.7075	0.9494	0	0.8462	1A
11.7216	1.3401	0.7294	0.6802	0.9375	0	0.8462	1A
13.1663	1.1875	0.7412	0.6995	0.9430	0	0.8462	1A
14.5717	1.2217	0.7459	0.6879	0.9238	0	0.8462	1A
10.1880	1.3251	0.7576	0.7019	0.9329	0	0.8462	1A
12.1267	1.2426	0.7318	0.6816	0.9323	0	0.8462	1A
16.2287	1.2080	0.7882	0.7447	0.9459	0	0.8462	1A
11.9053	1.2470	0.7506	0.6865	0.9175	0	0.8462	1A
10.4061	1.2501	0.7600	0.7279	0.9621	0	0.8462	1A
9.0901	1.2966	0.7388	0.6730	0.9132	0	0.8462	1A
10.6971	1.3245	0.7600	0.7170	0.9441	0	0.8462	1A
10.0657	1.3807	0.7482	0.6896	0.9238	0	0.8462	1A
11.6537	1.3789	0.7294	0.6958	0.9547	0	0.8462	1A
10.1224	1.2886	0.7529	0.7234	0.9623	0	0.8462	1A
11.2466	1.3654	0.7459	0.7129	0.9558	0	0.8462	1A
10.2101	1.2596	0.7576	0.7139	0.9438	0	0.8462	1A
10.4852	1.4571	0.7459	0.7014	0.9427	0	0.8462	1A
12.0845	1.3369	0.7176	0.6872	0.9603	0	0.8462	1A
13.6397	1.2590	0.7576	0.6998	0.9250	0	0.8462	1A
12.2961	1.2967	0.7482	0.6809	0.9114	0	0.8462	1A

FMT	NAS	PF	TPS	PS	CD	CA	COA
10.1355	1.3100	0.7365	0.6919	0.9419	0	0.8462	1A
9.9158	1.2275	0.7812	0.7494	0.9632	0	0.8462	1A
10.5636	1.3350	0.7553	0.6833	0.9082	0	0.8462	1A
11.2183	1.2345	0.7529	0.7176	0.9531	0	0.8462	1A
9.0979	1.3846	0.7506	0.7085	0.9462	0	0.8462	1A
52.9676	0.4146	0.9976	0.2864	0.2872	2	1	1B
80.2095	0.4394	0.9976	0.3554	0.3564	1	1	1B
78.4474	0.4263	0.9976	0.3102	0.311	4	1	1B
62.8528	0.5221	0.9976	0.3264	0.3273	5	1	1B
56.4942	0.4155	0.9976	0.3609	0.3618	4	1	1B
59.5435	0.4792	0.9976	0.3316	0.3324	3	1	1B
45.6509	0.5203	0.9388	0.2576	0.2743	2	1	1B
138.2842	0.4668	0.9976	0.3047	0.3056	4	1	1B
56.1804	0.4494	0.9976	0.3275	0.3283	2	1	1B
60.6415	0.4793	0.9976	0.297	0.2978	1	1	1B
57.9176	0.5018	0.9976	0.3248	0.3256	7	1	1B
87.6755	0.4578	0.9976	0.3046	0.3054	5	1	1B
80.3952	0.3371	0.9976	0.3386	0.3395	5	1	1B
83.4727	0.4538	0.9976	0.3736	0.3746	2	1	1B
58.8176	0.4889	0.9976	0.3342	0.3351	3	1	1B
87.2914	0.5395	0.9976	0.3295	0.3305	3	1	1B
58.3692	0.5467	0.9976	0.313	0.3138	1	1	1B
61.9579	0.6099	0.9976	0.3272	0.3281	2	1	1B
63.1433	0.5359	0.9976	0.3298	0.3307	1	1	1B
71.5462	0.5729	0.9976	0.3036	0.3043	1	1	1B
79.1065	0.4308	0.9976	0.3397	0.3407	3	1	1B
84.808	0.4637	0.9976	0.3111	0.312	0	1	1B
96.7954	0.4777	0.9976	0.32	0.3209	4	1	1B
71.5192	0.5009	0.9765	0.2974	0.3053	1	1	1B
85.5592	0.5556	0.9976	0.3246	0.3255	5	1	1B
82.6135	0.5347	0.9976	0.2693	0.2701	2	1	1B
71.8495	0.3883	0.9976	0.3054	0.3062	5	1	1B
110.0748	0.4061	0.9976	0.3246	0.3256	7	1	1B
84.9032	0.3738	0.9976	0.2964	0.2972	3	1	1B
62.1142	0.4921	0.9976	0.2834	0.2842	3	1	1B
87.7569	0.3322	0.9976	0.2912	0.292	4	1	1B
88.1566	0.3902	0.9976	0.3105	0.3113	2	1	1B
116.2587	0.5537	0.9976	0.3256	0.3265	2	1	1B
62.5021	0.4406	0.9976	0.2784	0.2791	5	1	1B
66.3734	0.355	0.9976	0.2984	0.2992	5	1	1B
80.6586	0.4661	0.9976	0.3411	0.342	0	1	1B
52.9527	0.3854	0.9788	0.33	0.3376	7	1	1B
85.2501	0.5998	0.9976	0.3667	0.3677	3	1	1B
76.4893	0.4631	0.9976	0.2693	0.2701	3	1	1B
71.104	0.3795	0.9976	0.2982	0.299	5	1	1B
90.275	0.4469	0.9976	0.3052	0.306	1	1	1B
58.6385	0.5545	0.9976	0.3436	0.3445	4	1	1B

FMT	NAS	PF	TPS	PS	CD	CA	COA
89.5289	0.3683	0.9976	0.3467	0.3476	2	1	1B
74.586	0.4158	0.9976	0.3244	0.3253	3	1	1B
86.7711	0.4184	0.9976	0.3484	0.3493	5	1	1B
80.6077	0.429	0.9976	0.2479	0.2486	2	1	1B
72.9093	0.5738	0.9976	0.3103	0.3112	4	1	1B
69.0219	0.5004	0.9976	0.2857	0.2865	6	1	1B
82.2343	0.4187	0.9976	0.3433	0.3443	6	1	1B
56.8961	0.4465	0.9976	0.3051	0.3059	2	1	1B
6.9691	4.3593	0.7412	0.6981	0.9427	0	0.8534	1C
7.492	4.3269	0.7529	0.7106	0.9438	0	0.8534	1C
8.3287	4.2787	0.7576	0.7335	0.9688	0	0.8534	1C
7.73	4.2723	0.8047	0.76	0.9444	0	0.8534	1C
7.6254	4.3809	0.7553	0.7241	0.9594	0	0.8534	1C
7.0711	4.3317	0.7553	0.7028	0.9312	0	0.8534	1C
7.0417	4.5191	0.7247	0.6824	0.9416	0	0.8534	1C
7.5678	4.2315	0.7694	0.7417	0.966	0	0.8534	1C
6.9281	4.4219	0.7365	0.7075	0.9615	0	0.8534	1C
7.1517	4.2804	0.7576	0.6974	0.9219	0	0.8534	1C
6.5345	4.4211	0.7435	0.6769	0.9111	0	0.8534	1C
7.6099	4.2192	0.7882	0.747	0.9489	0	0.8534	1C
7.3749	4.2793	0.7482	0.7224	0.9654	0	0.8534	1C
7.692	4.2146	0.7529	0.7059	0.9375	0	0.8534	1C
8.5518	4.3076	0.7647	0.7176	0.9385	0	0.8534	1C
7.9956	4.2436	0.7553	0.7153	0.947	0	0.8534	1C
7.7949	4.3674	0.7788	0.7365	0.9456	0	0.8534	1C
7.0248	4.4655	0.7459	0.7082	0.9495	0	0.8534	1C
6.72	4.3994	0.7341	0.7019	0.9553	0	0.8534	1C
7.5525	4.4536	0.7435	0.7166	0.9623	0	0.8534	1C
7.1184	4.2987	0.7435	0.7028	0.946	0	0.8534	1C
7.2369	4.212	0.7694	0.7264	0.9448	0	0.8534	1C
7.7485	4.1231	0.7929	0.7529	0.9496	0	0.8534	1C
7.0521	4.3496	0.7553	0.6901	0.913	0	0.8534	1C
6.8131	4.4426	0.7553	0.7123	0.9438	0	0.8534	1C
7.6545	4.2388	0.7459	0.695	0.9333	0	0.8534	1C
7.1541	4.2988	0.7294	0.6912	0.951	0	0.8534	1C
7.5723	4.138	0.7435	0.6768	0.9088	0	0.8534	1C
7.4578	4.2239	0.7482	0.7019	0.9373	0	0.8534	1C
8.022	4.28	0.7576	0.7275	0.9624	0	0.8534	1C
7.1484	4.2408	0.7318	0.6714	0.9167	0	0.8534	1C
8.0412	4.1846	0.7882	0.7358	0.9341	0	0.8534	1C
7.0445	4.2216	0.7529	0.7234	0.9623	0	0.8534	1C
7.665	4.2701	0.76	0.7153	0.9462	0	0.8534	1C
6.5345	4.2942	0.7412	0.6856	0.9265	0	0.8534	1C
6.9408	4.3248	0.76	0.7153	0.9412	0	0.8534	1C
7.61	4.3651	0.7482	0.7116	0.9525	0	0.8534	1C
7.3674	4.3905	0.7294	0.6792	0.932	0	0.8534	1C
6.9693	4.3318	0.7529	0.7075	0.9404	0	0.8534	1C

FMT	NAS	PF	TPS	PS	CD	CA	COA
7.5255	4.3723	0.7459	0.6948	0.9308	0	0.8534	1C
7.0469	4.2602	0.7576	0.7139	0.9438	0	0.8534	1C
6.7356	4.4528	0.7459	0.7055	0.9489	0	0.8534	1C
7.1947	4.3431	0.7176	0.6738	0.9406	0	0.8534	1C
6.7268	4.2482	0.7576	0.7106	0.9379	0	0.8534	1C
7.3305	4.2973	0.7506	0.6792	0.9057	0	0.8534	1C
7.1667	4.2633	0.7365	0.6918	0.9393	0	0.8534	1C
7.6553	4.2047	0.7812	0.7459	0.9548	0	0.8534	1C
6.7821	4.3192	0.76	0.7167	0.9465	0	0.8534	1C
8.5899	4.227	0.7553	0.716	0.9472	0	0.8534	1C
7.2642	4.3526	0.7506	0.7153	0.953	0	0.8534	1C
9.1293	3.5816	0.9976	0.7316	0.7333	1	1.0000	1D
10.4000	3.5670	0.9976	0.7500	0.7518	2	1.0000	1D
11.5310	3.5317	0.9976	0.7624	0.7642	6	1.0000	1D
10.8170	3.7213	0.9976	0.8147	0.8167	2	1.0000	1D
9.1434	3.6614	0.9976	0.7464	0.7482	1	1.0000	1D
10.1038	3.5877	0.9976	0.7405	0.7422	3	1.0000	1D
9.1750	3.7997	0.9976	0.7698	0.7716	1	1.0000	1D
12.3588	3.5160	0.9976	0.7512	0.7530	4	1.0000	1D
10.2462	3.6566	0.9976	0.7357	0.7375	3	1.0000	1D
9.6743	3.5250	0.9976	0.7630	0.7648	5	1.0000	1D
9.1120	3.6871	0.9976	0.7512	0.7529	2	1.0000	1D
12.0336	3.6020	0.9976	0.7780	0.7799	0	1.0000	1D
10.1343	3.5026	0.9976	0.7303	0.7321	1	1.0000	1D
11.7435	3.4305	0.9976	0.7387	0.7405	2	1.0000	1D
9.6917	3.6180	0.9976	0.7712	0.7730	1	1.0000	1D
11.4213	3.4371	0.9976	0.7547	0.7565	1	1.0000	1D
9.4550	3.7445	0.9976	0.7565	0.7583	0	1.0000	1D
9.6525	3.7297	0.9976	0.7547	0.7565	1	1.0000	1D
9.6765	3.6017	0.9976	0.7406	0.7423	2	1.0000	1D
8.9047	3.7459	0.9976	0.7394	0.7412	1	1.0000	1D
11.9245	3.5010	0.9976	0.7245	0.7262	3	1.0000	1D
10.7633	3.4918	0.9976	0.7464	0.7482	2	1.0000	1D
14.1308	3.3938	0.9976	0.7701	0.7720	0	1.0000	1D
9.7204	3.6753	0.9976	0.7482	0.7500	1	1.0000	1D
10.1042	3.7521	0.9976	0.7730	0.7749	2	1.0000	1D
11.4027	3.4233	0.9976	0.7530	0.7548	2	1.0000	1D
10.7875	3.4566	0.9976	0.7488	0.7506	2	1.0000	1D
12.6567	3.2913	0.9976	0.7571	0.7589	0	1.0000	1D
10.2746	3.3909	0.9976	0.7069	0.7085	3	1.0000	1D
9.6899	3.5797	0.9976	0.7305	0.7322	2	1.0000	1D
10.7698	3.3789	0.9976	0.7208	0.7225	4	1.0000	1D
13.0605	3.5320	0.9976	0.7589	0.7607	2	1.0000	1D
9.9304	3.4435	0.9976	0.7464	0.7482	1	1.0000	1D
10.5579	3.4686	0.9976	0.7373	0.7391	0	1.0000	1D
9.2348	3.4756	0.9976	0.7310	0.7327	1	1.0000	1D
9.9000	3.5798	0.9976	0.7352	0.7370	3	1.0000	1D

FMT	NAS	PF	TPS	PS	CD	CA	COA
9.4779	3.6320	0.9976	0.7423	0.7441	1	1.0000	1D
10.3573	3.5923	0.9976	0.7109	0.7126	0	1.0000	1D
9.7622	3.5653	0.9976	0.7441	0.7458	1	1.0000	1D
10.6510	3.5667	0.9976	0.7647	0.7665	4	1.0000	1D
10.4308	3.4840	0.9976	0.7452	0.7470	2	1.0000	1D
9.0162	3.7584	0.9976	0.7464	0.7482	3	1.0000	1D
11.6548	3.4457	0.9976	0.7314	0.7332	0	1.0000	1D
9.4646	3.5260	0.9976	0.7648	0.7667	2	1.0000	1D
9.2086	3.5424	0.9976	0.7167	0.7184	1	1.0000	1D
9.6773	3.4702	0.9976	0.7275	0.7292	1	1.0000	1D
10.2880	3.5374	0.9976	0.7705	0.7724	1	1.0000	1D
10.3546	3.5772	0.9976	0.7572	0.7590	1	1.0000	1D
9.7538	3.4982	0.9976	0.7441	0.7458	1	1.0000	1D
9.0496	3.6365	0.9976	0.7294	0.7311	4	1.0000	1D

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX 2: FCS RAW DATA

FMT	NAS	PF	TPS	PS	CD	CA	COA
3.4232	2.1049	0.8188	0.5165	0.6311	0	0.7414	2A
3.0793	2.1559	0.7718	0.5506	0.7134	0	0.7414	2A
3.4291	2.1111	0.7929	0.5399	0.6805	0	0.7414	2A
3.3145	2.1582	0.8024	0.5446	0.6784	0	0.7414	2A
3.1468	2.1416	0.8118	0.5529	0.6812	0	0.7414	2A
3.3493	2.1215	0.8071	0.5788	0.7172	0	0.7414	2A
3.0635	2.246	0.7576	0.533	0.704	0	0.7414	2A
3.3287	2.0942	0.7976	0.5504	0.6891	0	0.7414	2A
3.0974	2.1819	0.7812	0.564	0.7234	0	0.7414	2A
3.196	2.1166	0.7859	0.5495	0.6997	0	0.7414	2A
3.2167	2.2019	0.7718	0.5269	0.6818	0	0.7414	2A
3.3352	2.0903	0.8329	0.6203	0.745	0	0.7414	2A
3.105	2.099	0.8	0.5597	0.6988	0	0.7414	2A
3.3594	2.0035	0.8541	0.5761	0.674	0	0.7414	2A
3.3869	2.1245	0.7976	0.5318	0.6667	0	0.7414	2A
3.2715	2.0469	0.8376	0.5869	0.7003	0	0.7414	2A
3.1111	2.1997	0.7835	0.5446	0.6946	0	0.7414	2A
3.3444	2.1758	0.8047	0.554	0.688	0	0.7414	2A
3.2883	2.2052	0.7435	0.5446	0.7319	0	0.7414	2A
3.2754	2.1917	0.7953	0.5105	0.6412	0	0.7414	2A
3.3197	2.1496	0.7553	0.5647	0.7477	0	0.7414	2A
3.3659	2.1548	0.7506	0.561	0.7469	0	0.7414	2A
3.1913	2.1023	0.7671	0.5469	0.7125	0	0.7414	2A
3.1006	2.1643	0.7859	0.5493	0.6985	0	0.7414	2A
3.0779	2.2286	0.7435	0.4801	0.6447	0	0.7414	2A
3.5248	2.0871	0.7882	0.5647	0.7164	0	0.7414	2A
3.2471	2.1238	0.7718	0.5248	0.681	0	0.7414	2A
3.9056	2.0411	0.7788	0.5738	0.7357	0	0.7414	2A
3.35	2.0725	0.7882	0.5634	0.7143	0	0.7414	2A
3.2314	2.125	0.7882	0.5236	0.6647	0	0.7414	2A
3.4649	2.0416	0.8118	0.5493	0.6763	0	0.7414	2A
3.4069	2.0763	0.8235	0.5812	0.7057	0	0.7414	2A
3.1458	2.1542	0.7365	0.5236	0.7115	0	0.7414	2A
3.2994	2.0758	0.8094	0.545	0.6745	0	0.7414	2A
3.1009	2.1	0.7929	0.5647	0.7122	0	0.7414	2A
3.1138	2.1408	0.7812	0.5164	0.6607	0	0.7414	2A
3.2659	2.1484	0.7953	0.5529	0.6953	0	0.7414	2A
3.2595	2.1252	0.8094	0.5399	0.6667	0	0.7414	2A
3.2538	2.1516	0.7718	0.56	0.7256	0	0.7414	2A
3.312	2.1279	0.8165	0.5929	0.7262	0	0.7414	2A
3.6419	2.1242	0.7694	0.5412	0.7034	0	0.7414	2A
3.2606	2.1896	0.7835	0.5401	0.6898	0	0.7414	2A
3.2818	2.1012	0.7882	0.5461	0.6937	0	0.7414	2A
3.3709	2.0947	0.8047	0.5849	0.7273	0	0.7414	2A
3.4375	2.1321	0.7859	0.5236	0.6667	0	0.7414	2A

FMT	NAS	PF	TPS	PS	CD	CA	COA
3.2625	2.0979	0.7976	0.5976	0.7493	0	0.7414	2A
3.182	2.1239	0.7765	0.5236	0.6748	0	0.7414	2A
3.2852	2.1452	0.7765	0.5153	0.6636	0	0.7414	2A
3.3879	2.0808	0.8024	0.5718	0.7126	0	0.7414	2A
3.1565	2.159	0.7929	0.5376	0.6775	0	0.7414	2A
9.6613	1.2691	0.9976	0.2048	0.2053	4	1	2B
14.5224	1.215	0.9976	0.184	0.1844	6	1	2B
11.7831	1.1853	0.9976	0.1745	0.1749	6	1	2B
9.4254	1.3239	0.9976	0.2424	0.2429	4	1	2B
8.6008	1.3059	0.9976	0.208	0.2085	1	1	2B
10.2661	1.245	0.9976	0.228	0.2286	5	1	2B
9.1708	1.3808	0.9976	0.1943	0.1948	7	1	2B
11.9826	1.1678	0.9976	0.1844	0.1848	5	1	2B
8.544	1.3323	0.9976	0.1885	0.189	2	1	2B
9.1057	1.1754	0.9976	0.2043	0.2048	5	1	2B
8.5368	1.314	0.9976	0.1718	0.1722	6	1	2B
11.009	1.1913	0.9976	0.2214	0.222	1	1	2B
10.8566	1.1847	0.9976	0.1823	0.1827	3	1	2B
9.7578	1.1248	0.9976	0.2128	0.2133	3	1	2B
10.2408	1.2643	0.9976	0.1734	0.1738	6	1	2B
9.7044	1.1433	0.9976	0.1671	0.1675	4	1	2B
7.8368	1.3754	0.9976	0.2204	0.2209	4	1	2B
7.7933	1.3604	0.9976	0.1887	0.1891	4	1	2B
9.0284	1.2976	0.9976	0.1887	0.1891	3	1	2B
8.0753	1.35	0.9976	0.1925	0.1929	3	1	2B
9.801	1.1954	0.9976	0.2014	0.2019	3	1	2B
12.6654	1.1671	0.9976	0.1863	0.1868	3	1	2B
10.907	1.1154	0.9976	0.1679	0.1683	4	1	2B
10.0505	1.2916	0.9976	0.2235	0.2241	3	1	2B
8.0572	1.3414	0.9976	0.1801	0.1805	5	1	2B
12.1979	1.1457	0.9976	0.2134	0.2139	3	1	2B
9.0962	1.1588	0.9976	0.1818	0.1823	4	1	2B
12.6317	1.0305	0.9976	0.2047	0.2052	2	1	2B
10.8304	1.1223	0.9976	0.2304	0.231	5	1	2B
9.0401	1.2369	0.9976	0.1934	0.1939	3	1	2B
13.6755	1.1075	0.9976	0.1816	0.182	4	1	2B
11.6641	1.1749	0.9976	0.191	0.1915	3	1	2B
10.3065	1.1316	0.9976	0.1967	0.1971	2	1	2B
10.8157	1.2093	0.9976	0.1699	0.1703	5	1	2B
11.8751	1.1609	0.9976	0.1971	0.1976	2	1	2B
9.6545	1.2754	0.9976	0.1698	0.1702	7	1	2B
8.4899	1.2832	0.9976	0.1991	0.1995	2	1	2B
9.7525	1.2409	0.9976	0.227	0.2275	4	1	2B
9.0193	1.2372	0.9976	0.2156	0.2162	1	1	2B
10.2839	1.2625	0.9976	0.2306	0.2311	2	1	2B
10.512	1.1369	0.9976	0.181	0.1814	2	1	2B
10.6253	1.3631	0.9976	0.1855	0.186	5	1	2B

FMT	NAS	PF	TPS	PS	CD	CA	COA
9.8558	1.1949	0.9976	0.1548	0.1551	4	1	2B
11.0848	1.1786	0.9976	0.1726	0.173	5	1	2B
11.8934	1.1896	0.9976	0.2241	0.2246	2	1	2B
10.255	1.1698	0.9976	0.2019	0.2024	3	1	2B
8.4128	1.2181	0.9976	0.1779	0.1783	1	1	2B
10.94	1.2577	0.9976	0.2225	0.223	2	1	2B
10.9993	1.1629	0.9976	0.2067	0.2071	5	1	2B
8.224	1.3251	0.9976	0.169	0.1695	4	1	2B
2.9554	5.1027	0.8212	0.5282	0.6429	0	0.7478	2C
2.8873	5.1343	0.7835	0.5459	0.6967	0	0.7478	2C
2.9975	5.1059	0.8	0.5211	0.651	0	0.7478	2C
2.96	5.1383	0.8141	0.5728	0.7032	0	0.7478	2C
2.838	5.126	0.8259	0.5141	0.6222	0	0.7478	2C
2.9695	5.1028	0.8235	0.5873	0.7135	0	0.7478	2C
2.8613	5.2233	0.7718	0.4929	0.6391	0	0.7478	2C
2.9569	5.077	0.8165	0.5469	0.6695	0	0.7478	2C
2.9513	5.1727	0.7859	0.5637	0.7177	0	0.7478	2C
2.9714	5.098	0.8	0.5354	0.6696	0	0.7478	2C
2.9434	5.1773	0.7859	0.5152	0.6548	0	0.7478	2C
3.0847	5.0751	0.84	0.6014	0.7163	0	0.7478	2C
2.8595	5.0662	0.8188	0.5738	0.7	0	0.7478	2C
2.9827	4.999	0.8588	0.541	0.6294	0	0.7478	2C
3.0386	5.104	0.8165	0.5377	0.659	0	0.7478	2C
3.0399	5.0367	0.8447	0.6009	0.7111	0	0.7478	2C
2.8708	5.19	0.7929	0.5822	0.7337	0	0.7478	2C
3.0122	5.1479	0.8259	0.5352	0.6477	0	0.7478	2C
2.9151	5.1919	0.7506	0.5	0.6656	0	0.7478	2C
2.9323	5.1839	0.8071	0.5423	0.6715	0	0.7478	2C
3.0615	5.1251	0.7741	0.5283	0.6829	0	0.7478	2C
2.8289	5.143	0.76	0.5446	0.716	0	0.7478	2C
2.9554	5.0656	0.7882	0.5728	0.7262	0	0.7478	2C
2.9036	5.1421	0.8118	0.5563	0.685	0	0.7478	2C
3.0008	5.2112	0.7553	0.5199	0.6873	0	0.7478	2C
3.138	5.0581	0.8071	0.5694	0.7055	0	0.7478	2C
2.9341	5.1025	0.7882	0.5296	0.6727	0	0.7478	2C
3.0917	5.0228	0.7953	0.541	0.6794	0	0.7478	2C
2.9684	5.0654	0.7929	0.5657	0.713	0	0.7478	2C
2.9835	5.1153	0.7929	0.5448	0.6875	0	0.7478	2C
2.9841	5.0265	0.8212	0.5516	0.6714	0	0.7478	2C
2.9824	5.0673	0.8353	0.5704	0.6826	0	0.7478	2C
2.8129	5.1439	0.7506	0.5354	0.7138	0	0.7478	2C
3.0551	5.0482	0.8259	0.5782	0.7011	0	0.7478	2C
3.0024	5.0869	0.8047	0.5788	0.7193	0	0.7478	2C
2.8538	5.155	0.7812	0.5305	0.6787	0	0.7478	2C
2.9129	5.1307	0.8165	0.5812	0.7118	0	0.7478	2C
2.8842	5.1079	0.8235	0.5469	0.6638	0	0.7478	2C
2.9341	5.1263	0.7882	0.5412	0.6866	0	0.7478	2C

FMT	NAS	PF	TPS	PS	CD	CA	COA
2.9137	5.1132	0.8212	0.5751	0.7	0	0.7478	2C
2.8966	5.1063	0.7835	0.5129	0.6547	0	0.7478	2C
2.9744	5.175	0.7953	0.5542	0.6973	0	0.7478	2C
3.0675	5.0836	0.8	0.5461	0.6834	0	0.7478	2C
2.935	5.071	0.8188	0.5236	0.6398	0	0.7478	2C
2.8272	5.1135	0.8047	0.5211	0.6472	0	0.7478	2C
2.8883	5.0791	0.8141	0.5647	0.6936	0	0.7478	2C
2.9564	5.0979	0.7882	0.5141	0.6518	0	0.7478	2C
3.0022	5.139	0.7859	0.5576	0.7096	0	0.7478	2C
2.8994	5.066	0.8094	0.5129	0.6337	0	0.7478	2C
2.8379	5.1557	0.8024	0.5352	0.6667	0	0.7478	2C
3.7588	4.8069	0.9741	0.5835	0.599	1	1	2D
3.4789	4.7719	0.9906	0.5788	0.5843	2	1	2D
3.4735	4.7876	0.9788	0.5657	0.5779	0	1	2D
3.8007	4.8285	0.9906	0.5812	0.5867	3	1	2D
3.51	4.8097	0.9882	0.5981	0.6053	1	1	2D
3.5406	4.7746	0.9859	0.5708	0.5789	1	1	2D
3.696	4.845	0.9788	0.5637	0.5759	2	1	2D
3.6772	4.7752	0.9788	0.5906	0.6034	0	1	2D
3.5899	4.8433	0.9741	0.5896	0.6053	0	1	2D
3.652	4.7281	0.9835	0.5532	0.5625	4	1	2D
3.6699	4.8576	0.9788	0.5915	0.6043	1	1	2D
3.65	4.7465	0.9788	0.5626	0.5749	1	1	2D
3.3757	4.7804	0.9812	0.6226	0.6346	0	1	2D
3.6134	4.7194	0.9882	0.6455	0.6532	0	1	2D
3.7842	4.7993	0.9788	0.6329	0.6466	0	1	2D
3.8397	4.7077	0.9882	0.6	0.6071	1	1	2D
3.6795	4.8586	0.9812	0.6197	0.6316	0	1	2D
3.5925	4.8546	0.9835	0.5849	0.5947	0	1	2D
3.4133	4.848	0.9741	0.6085	0.6247	0	1	2D
3.5963	4.8895	0.9694	0.5563	0.5738	2	1	2D
3.5736	4.724	0.9835	0.5283	0.5372	1	1	2D
3.5786	4.7092	0.9882	0.5953	0.6024	2	1	2D
3.7691	4.6476	0.9788	0.5235	0.5348	2	1	2D
3.493	4.8323	0.9788	0.6009	0.6139	2	1	2D
3.4228	4.8737	0.9812	0.6071	0.6187	0	1	2D
3.7033	4.6742	0.9741	0.5354	0.5496	2	1	2D
3.5499	4.7487	0.9835	0.6043	0.6145	1	1	2D
3.7007	4.6675	0.9788	0.6009	0.6139	1	1	2D
3.5518	4.6937	0.9741	0.6033	0.6193	1	1	2D
3.5995	4.7868	0.9812	0.5943	0.6058	0	1	2D
3.7848	4.6869	0.9718	0.5587	0.5749	2	1	2D
3.7498	4.7653	0.9812	0.5835	0.5947	2	1	2D
3.5665	4.7174	0.9859	0.5519	0.5598	2	1	2D
3.5218	4.7339	0.9812	0.5972	0.6087	0	1	2D
3.8332	4.7273	0.9741	0.6085	0.6247	1	1	2D
3.485	4.805	0.9741	0.6094	0.6256	0	1	2D

FMT	NAS	PF	TPS	PS	CD	CA	COA
3.5289	4.8139	0.9718	0.5459	0.5617	3	1	2D
3.464	4.8092	0.9788	0.5553	0.5673	1	1	2D
3.6728	4.773	0.9694	0.5613	0.5791	0	1	2D
3.6984	4.8241	0.9647	0.5446	0.5645	1	1	2D
3.7763	4.6909	0.9859	0.5859	0.5943	3	1	2D
3.6571	4.8935	0.9812	0.6123	0.6241	0	1	2D
3.5098	4.7184	0.9882	0.6052	0.6124	0	1	2D
3.6817	4.7436	0.9741	0.5495	0.5642	1	1	2D
3.6338	4.7682	0.9765	0.6118	0.6265	1	1	2D
3.7752	4.7118	0.9694	0.5647	0.5825	2	1	2D
3.483	4.7566	0.9835	0.6108	0.6211	1	1	2D
3.4327	4.7743	0.9882	0.5498	0.5564	0	1	2D
3.9431	4.7202	0.9765	0.5731	0.587	0	1	2D
3.6601	4.8122	0.9765	0.554	0.5673	0	1	2D

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX 3: MAUT FUNCTION

```
function(weights, data)
{
#
# strip COA designator from data
        temp <- data[, 1:7]        # get best and worst column data
        bestFMT <- min(temp[, 1])
        worstFMT <- max(temp[, 1])
        bestNAS <- max(temp[, 2])
        worstNAS <- min(temp[, 2])
        bestPF <- max(temp[, 3])
        worstPF <- min(temp[, 3])
        bestTPS <- max(temp[, 4])
        worstTPS <- min(temp[, 4])
        bestPS <- max(temp[, 5])
        worstPS <- min(temp[, 5])
        bestCD <- min(temp[, 6])
        worstCD <- max(temp[, 6])
        bestCA <- max(temp[, 7])
        worstCA <- min(temp[, 7])
        best <- c(bestFMT, bestNAS, bestPF, bestTPS, bestPS, bestCD, bestCA)
        worst <- c(worstFMT, worstNAS, worstPF, worstTPS, worstPS, worstCD, worstCA)
        # compute individualvalues for each column
        difference <- best - worst
        for(i in 1:length(temp[, 1])) {
                temp[i,  ] <- (temp[i,  ] - worst)/difference
        }
# mult columns by weight and then sum the rows to get MAUT values
        x <- t(weights %*% t(temp))
        x <- data.frame(x, data[, 8])
        names(x) <- c("UTILITY", "COA")
        return(x)
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX 4: FSST PROGRAM

A. INSTANCE METHODS

1. AO

```
/**
 * Juan K. Ulloa <BR>
 * Thesis: FSST
 * January 13, 2001
 *
 * <P> This class establishes the Area of Operations (AO) for the
 *     simulation.  This class also establishes the parameters for
 *     what distributions dictate where targets arrive in the AO.
**/

package fsst;
import simkit.*;
import simkit.smd.*;
import simkit.data.*;

public class AO{

// class variables
   public static RandomNumber seed;

// instance variables
   private RandomVariate xCoord;
   private RandomVariate yCoord;
   private Coordinate lowerLeft;
   private Coordinate upperRight;
   private String xDistribution;
   private String yDistribution;
   private Object[] xDistributionParameters;
   private Object[] yDistributionParameters;

// constructors
   static {seed = RandomFactory.getRandomNumber();}

   public AO(Coordinate theLowerLeft, Coordinate theUpperRight,
     String xDist, Object[] xDistParams, String yDist, Object[] yDistParams){
      xDistribution = xDist;
      xDistributionParameters = (Object[])xDistParams.clone();
      yDistribution = yDist;
      yDistributionParameters = (Object[])yDistParams.clone();
      lowerLeft = new Coordinate(theLowerLeft);
      upperRight = new Coordinate(theUpperRight);
      xCoord = RandomFactory.getRandomVariate(xDistribution, xDistributionParameters,
        RandomFactory.getRandomNumber());
      yCoord = RandomFactory.getRandomVariate(yDistribution, yDistributionParameters,
        RandomFactory.getRandomNumber());
    }

    public AO(AO theBox){
      this(theBox.getLowerLeft(),theBox.getUpperRight(), theBox.getXDistribution(),
        theBox.getXDistributionParameters(), theBox.getYDistribution(),
        theBox.getYDistributionParameters());
    }

// class methods
   public static void setSeed(long theSeed){
       seed.setSeed(theSeed);
   }

// instance methods
   public Coordinate getLowerLeft(){return new Coordinate(lowerLeft);}

   public Coordinate getUpperRight(){return new Coordinate(upperRight);}
   public Coordinate getRandomLocation(){
      double xCoordinate = xCoord.generate();
      double yCoordinate = upperRight.getYCoord()-yCoord.generate();
      if ((xCoordinate >= lowerLeft.getXCoord()) &&
        (xCoordinate <= upperRight.getXCoord()) &&
        (yCoordinate >= 0)){
         return new Coordinate(xCoordinate, yCoordinate);
      }
      else {
         return this.getRandomLocation();
      }
   }

   public String getXDistribution(){return new String(xDistribution);}

   public String getYDistribution(){return new String(yDistribution);}

   public Object[] getXDistributionParameters(){
      return (Object[])xDistributionParameters.clone();
   }

   public Object[] getYDistributionParameters(){
      return (Object[])yDistributionParameters.clone();
   }

   public String toString(){
      return "Lower left: " + this.getLowerLeft() + ", Upper Right: " +
        this.getUpperRight() + ", X distribution: " + this.getXDistribution() +
        ", X distribution parameters: " + this.getXDistributionParameters() +
        ", Y distribution: " + this.getYDistribution() +
        ", Y distribution parameters: " + this.getYDistributionParameters();
   }

}
```

2. BattlefieldData

```
/**
 * Juan K. Ulloa <BR>
 * Thesis: FSST
 * January 13, 2001
 *
 * <P> This class tracks the battlefield data such as makeup, etc.
 *
**/

package fsst;
import java.util.*;

public class BattleFieldData{

   private double percentArmor;
   private double percentInfantryInOpen;
   private double percentInfantryDugIn;
   private double percentArmoredPC;
   private double percentLightSkinVehicle;

   private int destroyArmor;
   private int destroyArmorSalvo;
   private int neutralizeArmor;
   private int neutralizeArmorSalvo;
   private int suppressArmor;
   private int suppressArmorSalvo;

   private int destroyIIO;
   private int destroyIIOSalvo;
   private int neutralizeIIO;
   private int neutralizeIIOSalvo;
   private int suppressIIO;
   private int suppressIIOSalvo;
   private int destroyIDI;
   private int destroyIDISalvo;
   private int neutralizeIDI;
   private int neutralizeIDISalvo;
   private int suppressIDI;
   private int suppressIDISalvo;

   private int destroyAPC;
   private int destroyAPCSalvo;
   private int neutralizeAPC;
   private int neutralizeAPCSalvo;
   private int suppressAPC;
   private int suppressAPCSalvo;

   private int destroyLSV;
   private int destroyLSVSalvo;
   private int neutralizeLSV;
   private int neutralizeLSVSalvo;
   private int suppressLSV;
   private int suppressLSVSalvo;

   private double percentDestroy;
   private double percentNeutralize;
   private double percentSuppress;

   private double fleeProbability;
   private long seed;
   private Random r;

   private double collateralDamageRadius;
   private double collateralDamagePercent;
   private int numberCollateralDamageCausingRounds;

   public BattleFieldData(double thePercentArmor,int theDestroyArmor,
     int theDestroyArmorSalvo,int theNeutralizeArmor,int theNeutralizeArmorSalvo,
     int theSuppressArmor,int theSuppressArmorSalvo,double thePercentInfantryInOpen,
     int theDestroyIIO,int theDestroyIIOSalvo,int theNeutralizeIIO,
     int theNeutralizeIIOSalvo,int theSuppressIIO,int theSuppressIIOSalvo,
     double thePercentInfantryDugIn,int theDestroyIDI,int theDestroyIDISalvo,
     int theNeutralizeIDI,int theNeutralizeIDISalvo,int theSuppressIDI,
     int theSuppressIDISalvo,double thePercentArmoredPC,int theDestroyAPC,
     int theDestroyAPCSalvo,int theNeutralizeAPC,int theNeutralizeAPCSalvo,
     int theSuppressAPC,int theSuppressAPCSalvo,double thePercentLightSkinVehicle,
     int theDestroyLSV,int theDestroyLSVSalvo,int theNeutralizeLSV,
     int theNeutralizeLSVSalvo,int theSuppressLSV,int theSuppressLSVSalvo,
     double thePercentDestroy, double thePercentNeutralize, double thePercentSuppress,
     double theFleeProbability,long theSeed,double theCollateralDamageRadius,
     double theCollateralDamagePercent){

      percentArmor = thePercentArmor;
      destroyArmor = theDestroyArmor;
      destroyArmorSalvo = theDestroyArmorSalvo;
      neutralizeArmor = theNeutralizeArmor;
      neutralizeArmorSalvo = theNeutralizeArmorSalvo;
      suppressArmor = theSuppressArmor;
      suppressArmorSalvo = theSuppressArmorSalvo;

      percentInfantryInOpen = thePercentInfantryInOpen;
      destroyIIO = theDestroyIIO;
      destroyIIOSalvo = theDestroyIIOSalvo;
      neutralizeIIO = theNeutralizeIIO;
      neutralizeIIOSalvo = theNeutralizeIIOSalvo;
      suppressIIO = theSuppressIIO;
      suppressIIOSalvo = theSuppressIIOSalvo;

      percentInfantryDugIn = thePercentInfantryDugIn;
      destroyIDI = theDestroyIDI;
      destroyIDISalvo = theDestroyIDISalvo;
      neutralizeIDI = theNeutralizeIDI;
      neutralizeIDISalvo = theNeutralizeIDISalvo;
```

```
        suppressIDI = theSuppressIDI;
        suppressIDISalvo = theSuppressIDISalvo;

        percentArmoredPC = thePercentArmoredPC;
        destroyAPC = theDestroyAPC;
        destroyAPCSalvo = theDestroyAPCSalvo;
        neutralizeAPC = theNeutralizeAPC;
        neutralizeAPCSalvo = theNeutralizeAPCSalvo;
        suppressAPC = theSuppressAPC;
        suppressAPCSalvo = theSuppressAPCSalvo;

        percentLightSkinVehicle = thePercentLightSkinVehicle;
        destroyLSV = theDestroyLSV;
        destroyLSVSalvo = theDestroyLSVSalvo;
        neutralizeLSV = theNeutralizeLSV;
        neutralizeLSVSalvo = theNeutralizeLSVSalvo;
        suppressLSV = theSuppressLSV;
        suppressLSVSalvo = theSuppressLSVSalvo;

        percentDestroy = thePercentDestroy;
        percentNeutralize = thePercentNeutralize;
        percentSuppress = thePercentSuppress;

        setSeed(theSeed);
        r = new Random(theSeed);
        setFleeProbability(theFleeProbability);
        collateralDamageRadius = theCollateralDamageRadius;
        collateralDamagePercent = theCollateralDamagePercent;
    }

    public BattleFieldData(BattleFieldData theBattleFieldData){
        percentArmor = theBattleFieldData.getPercentArmor();
        destroyArmor = theBattleFieldData.getDestroyArmor();
        destroyArmorSalvo = theBattleFieldData.getDestroyArmorSalvo();
        neutralizeArmor = theBattleFieldData.getNeutralizeArmor();
        neutralizeArmorSalvo = theBattleFieldData.getNeutralizeArmorSalvo();
        suppressArmor = theBattleFieldData.getSuppressArmor();
        suppressArmorSalvo = theBattleFieldData.getSuppressArmorSalvo();

        percentInfantryInOpen = theBattleFieldData.getPercentIIO();
        destroyIIO = theBattleFieldData.getDestroyIIO();
        destroyIIOSalvo = theBattleFieldData.getDestroyIIOSalvo();
        neutralizeIIO = theBattleFieldData.getNeutralizeIIO();
        neutralizeIIOSalvo = theBattleFieldData.getNeutralizeIIOSalvo();
        suppressIIO = theBattleFieldData.getSuppressIIO();
        suppressIIOSalvo = theBattleFieldData.getSuppressIIOSalvo();

        percentInfantryDugIn = theBattleFieldData.getPercentIDI();
        destroyIDI = theBattleFieldData.getDestroyIDI();
        destroyIDISalvo = theBattleFieldData.getDestroyIDISalvo();
        neutralizeIDI = theBattleFieldData.getNeutralizeIDI();
        neutralizeIDISalvo = theBattleFieldData.getNeutralizeIDISalvo();
        suppressIDI = theBattleFieldData.getSuppressIDI();
        suppressIDISalvo = theBattleFieldData.getSuppressIDISalvo();

        percentArmoredPC = theBattleFieldData.getPercentAPC();
        destroyAPC = theBattleFieldData.getDestroyAPC();
        destroyAPCSalvo = theBattleFieldData.getDestroyAPCSalvo();
        neutralizeAPC = theBattleFieldData.getNeutralizeAPC();
        neutralizeAPCSalvo = theBattleFieldData.getNeutralizeAPCSalvo();
        suppressAPC = theBattleFieldData.getSuppressAPC();
        suppressAPCSalvo = theBattleFieldData.getSuppressAPCSalvo();

        percentLightSkinVehicle = theBattleFieldData.getPercentLSV();
        destroyLSV = theBattleFieldData.getDestroyLSV();
        destroyLSVSalvo = theBattleFieldData.getDestroyLSVSalvo();
        neutralizeLSV = theBattleFieldData.getNeutralizeLSV();
        neutralizeLSVSalvo = theBattleFieldData.getNeutralizeLSVSalvo();
        suppressLSV = theBattleFieldData.getSuppressLSV();
        suppressLSVSalvo = theBattleFieldData.getSuppressLSVSalvo();
    percentDestroy = theBattleFieldData.getPercentDestroy();
    percentNeutralize = theBattleFieldData.getPercentNeutralize();
    percentSuppress = theBattleFieldData.getPercentSuppress();

    setSeed(theBattleFieldData.getSeed());
    r = new Random(seed);
    setFleeProbability(theBattleFieldData.getFleeProbability());
    collateralDamageRadius = theBattleFieldData.getCollateralDamageRadius();
    collateralDamagePercent = theBattleFieldData.getCollateralDamagePercent();
}

public double getPercentArmor(){return percentArmor;}

public int getDestroyArmor(){return destroyArmor;}

public int getDestroyArmorSalvo(){return destroyArmorSalvo;}

public int getNeutralizeArmor(){return neutralizeArmor;}

public int getNeutralizeArmorSalvo(){return neutralizeArmorSalvo;}

public int getSuppressArmor(){return suppressArmor;}

public int getSuppressArmorSalvo(){return suppressArmorSalvo;}

public double getPercentIIO(){return percentInfantryInOpen;}

public int getDestroyIIO(){return destroyIIO;}

public int getDestroyIIOSalvo(){return destroyIIOSalvo;}

public int getNeutralizeIIO(){return neutralizeIIO;}

public int getNeutralizeIIOSalvo(){return neutralizeIIOSalvo;}

public int getSuppressIIO(){return suppressIIO;}

public int getSuppressIIOSalvo(){return suppressIIOSalvo;}

public double getPercentIDI(){return percentInfantryDugIn;}

public int getDestroyIDI(){return destroyIDI;}

public int getDestroyIDISalvo(){return destroyIDISalvo;}

public int getNeutralizeIDI(){return neutralizeIDI;}

public int getNeutralizeIDISalvo(){return neutralizeIDISalvo;}

public int getSuppressIDI(){return suppressIDI;}

public int getSuppressIDISalvo(){return suppressIDISalvo;}

public double getPercentAPC(){return percentArmoredPC;}

public int getDestroyAPC(){return destroyAPC;}

public int getDestroyAPCSalvo(){return destroyAPCSalvo;}

public int getNeutralizeAPC(){return neutralizeAPC;}

public int getNeutralizeAPCSalvo(){return neutralizeAPCSalvo;}

public int getSuppressAPC(){return suppressAPC;}

public int getSuppressAPCSalvo(){return suppressAPCSalvo;}

public double getPercentLSV(){return percentLightSkinVehicle;}

public int getDestroyLSV(){return destroyLSV;}
public int getDestroyLSVSalvo(){return destroyLSVSalvo;}

public int getNeutralizeLSV(){return neutralizeLSV;}

public int getNeutralizeLSVSalvo(){return neutralizeLSVSalvo;}

public int getSuppressLSV(){return suppressLSV;}

public int getSuppressLSVSalvo(){return suppressLSVSalvo;}

public double getPercentDestroy(){return percentDestroy;}

public double getPercentNeutralize(){return percentNeutralize;}

public double getPercentSuppress(){return percentSuppress;}

public boolean isDestroyed(String targetType, int hits){
   if (targetType.equals("Armor")){
      return (hits>getDestroyArmor());
   }
   else if(targetType.equals("InfantryInOpen")){
      return (hits>getDestroyIIO());
   }
   else if(targetType.equals("InfantryDugIn")){
      return (hits>getDestroyIDI());
   }
   else if(targetType.equals("APC")){
      return (hits>getDestroyAPC());
   }
   else {
      return (hits>getDestroyLSV());
   }
}

public boolean isNeutralized(String targetType, int hits){
   if (targetType.equals("Armor")){
      return (hits>getNeutralizeArmor());
   }
   else if(targetType.equals("InfantryInOpen")){
      return (hits>getNeutralizeIIO());
   }
   else if(targetType.equals("InfantryDugIn")){
      return (hits>getNeutralizeIDI());
   }
   else if(targetType.equals("APC")){
      return (hits>getNeutralizeAPC());
   }
   else {
      return (hits>getNeutralizeLSV());
   }
}

public boolean isSuppressed(String targetType, int closeCalls){
   if (targetType.equals("Armor")){
      return (closeCalls>getSuppressArmor());
   }
   else if(targetType.equals("InfantryInOpen")){
      return (closeCalls>getSuppressIIO());
   }
   else if(targetType.equals("InfantryDugIn")){
      return (closeCalls>getSuppressIDI());
   }
   else if(targetType.equals("APC")){
      return (closeCalls>getSuppressAPC());
   }
   else {
      return (closeCalls>getSuppressLSV());
   }
}

public double getFleeProbability(){return fleeProbability;}
```

```
   public void setFleeProbability(double theFleeProbability){
      fleeProbability = theFleeProbability;
   }

   public double getCollateralDamageRadius(){return collateralDamageRadius;}

   public double getCollateralDamagePercent(){return collateralDamagePercent;}

   public boolean didFlee(){return (r.nextDouble()<fleeProbability);}

   public long getSeed(){return seed;}

   public void setSeed(long theSeed){
      seed = theSeed;
   }

   public String getTargetType(){
      double targetGenerator = r.nextDouble();
      if (targetGenerator <= percentArmor){
         return "Armor";
      }
      else if (targetGenerator <= (percentArmor+percentInfantryInOpen)){
         return "InfantryInOpen";
      }
      else if (targetGenerator <= (percentArmor+percentInfantryInOpen+
        percentInfantryDugIn)){
         return "InfantryDugIn";
      }
      else if (targetGenerator <= (percentArmor+percentInfantryInOpen+
        percentInfantryDugIn + percentArmoredPC)){
         return "APC";
      }
      else{
         return "LightSkinVehicle";
      }
   }

   public String getFireMissionType(){
      double missionGenerator = r.nextDouble();
      if (missionGenerator <= percentDestroy){
         return "Destroy";
      }
      else if (missionGenerator <= (percentDestroy+percentNeutralize)){
         return "Neutralize";
      }
      else{
         return "Suppress";
      }
   }

   public int getSalvoSize(String theTargetType, String theMission){
      if ((theTargetType.equals("Armor"))&&(theMission.equals("Destroy"))){
         return destroyArmorSalvo;
      }
      else if ((theTargetType.equals("Armor"))&&(theMission.equals("Neutralize"))){
         return neutralizeArmorSalvo;
      }
      else if ((theTargetType.equals("Armor"))&&(theMission.equals("Suppress"))){
         return suppressArmorSalvo;
      }
      else if ((theTargetType.equals("InfantryInOpen"))&&(theMission.equals("Destroy"))){
         return destroyIIOSalvo;
      }
      else if
((theTargetType.equals("InfantryInOpen"))&&(theMission.equals("Neutralize"))){
         return neutralizeIIOSalvo;
      }
      else if
((theTargetType.equals("InfantryInOpen"))&&(theMission.equals("Suppress"))){
         return suppressIIOSalvo;
      }
      else if ((theTargetType.equals("InfantryDugIn"))&&(theMission.equals("Destroy"))){
         return destroyIDISalvo;
      }
      else if
((theTargetType.equals("InfantryDugIn"))&&(theMission.equals("Neutralize"))){
         return neutralizeIDISalvo;
      }
      else if ((theTargetType.equals("InfantryDugIn"))&&(theMission.equals("Suppress"))){
         return suppressIDISalvo;
      }
      else if ((theTargetType.equals("APC"))&&(theMission.equals("Destroy"))){
         return destroyAPCSalvo;
      }
      else if ((theTargetType.equals("APC"))&&(theMission.equals("Neutralize"))){
         return neutralizeAPCSalvo;
      }
      else if ((theTargetType.equals("APC"))&&(theMission.equals("Suppress"))){
         return suppressAPCSalvo;
      }
      else if
((theTargetType.equals("LightSkinVehicle"))&&(theMission.equals("Destroy"))){
         return destroyLSVSalvo;
      }
      else if
((theTargetType.equals("LightSkinVehicle"))&&(theMission.equals("Neutralize"))){
         return neutralizeLSVSalvo;
      }
      else{
         return suppressLSVSalvo;
      }
   }

   public int getHitsNeeded(String theTargetType, String theMission){
      if ((theTargetType.equals("Armor"))&&(theMission.equals("Destroy"))){
         return destroyArmor;
      }
      else if ((theTargetType.equals("Armor"))&&(theMission.equals("Neutralize"))){
         return neutralizeArmor;
      }
      else if ((theTargetType.equals("Armor"))&&(theMission.equals("Suppress"))){
         return suppressArmor;
      }
      else if ((theTargetType.equals("InfantryInOpen"))&&(theMission.equals("Destroy"))){
         return destroyIIO;
      }
      else if
((theTargetType.equals("InfantryInOpen"))&&(theMission.equals("Neutralize"))){
         return neutralizeIIO;
      }
      else if
((theTargetType.equals("InfantryInOpen"))&&(theMission.equals("Suppress"))){
         return suppressIIO;
      }
      else if ((theTargetType.equals("InfantryDugIn"))&&(theMission.equals("Destroy"))){
         return destroyIDI;
      }
      else if
((theTargetType.equals("InfantryDugIn"))&&(theMission.equals("Neutralize"))){
         return neutralizeIDI;
      }
      else if ((theTargetType.equals("InfantryDugIn"))&&(theMission.equals("Suppress"))){
         return suppressIDI;
      }
      else if ((theTargetType.equals("APC"))&&(theMission.equals("Destroy"))){
         return destroyAPC;
      }
      else if ((theTargetType.equals("APC"))&&(theMission.equals("Neutralize"))){
         return neutralizeAPC;
      }
      else if ((theTargetType.equals("APC"))&&(theMission.equals("Suppress"))){
         return suppressAPC;
      }
      else if
((theTargetType.equals("LightSkinVehicle"))&&(theMission.equals("Destroy"))){
         return destroyLSV;
      }
      else if
((theTargetType.equals("LightSkinVehicle"))&&(theMission.equals("Neutralize"))){
         return neutralizeLSV;
      }
      else{
         return suppressLSV;
      }
   }

   public int getDestroy(String theTargetType){
      if (theTargetType.equals("Armor")){
         return destroyArmor;
      }
      else if (theTargetType.equals("InfantryInOpen")){
         return destroyIIO;
      }
      else if (theTargetType.equals("InfantryDugIn")){
         return destroyIDI;
      }
      else if (theTargetType.equals("APC")){
         return destroyAPC;
      }
      else{
         return destroyLSV;
      }
   }

   public int getNeutralize(String theTargetType){
      if (theTargetType.equals("Armor")){
         return neutralizeArmor;
      }
      else if (theTargetType.equals("InfantryInOpen")){
         return neutralizeIIO;
      }
      else if (theTargetType.equals("InfantryDugIn")){
         return neutralizeIDI;
      }
      else if (theTargetType.equals("APC")){
         return neutralizeAPC;
      }
      else{
         return neutralizeLSV;
      }
   }

   public int getSuppress(String theTargetType){
      if (theTargetType.equals("Armor")){
         return suppressArmor;
      }
      else if (theTargetType.equals("InfantryInOpen")){
         return suppressIIO;
      }
      else if (theTargetType.equals("InfantryDugIn")){
         return suppressIDI;
      }
      else if (theTargetType.equals("APC")){
         return suppressAPC;
      }
      else{
         return suppressLSV;
      }
   }

   public void increaseNumberCollateralDamageCausingRounds(){
      numberCollateralDamageCausingRounds++;
   }

   public int getNumberCollateralDamageCausingRounds(){
      return numberCollateralDamageCausingRounds;
   }

   public void resetCollateralDamageCausingRounds(){
      numberCollateralDamageCausingRounds=0;
   }
   public double getRandomNumber(){return r.nextDouble();}

}
```

3. Shooter

```
/**
 * Juan K. Ulloa <BR>
 * Thesis: FSST
 * January 13, 2001
 *
 * <P> This class creates an instance of an artillery piece (shooter).
 *
**/

package fsst;
import simkit.*;
import simkit.data.*;
import simkit.smd.*;
import java.text.DecimalFormat;

public class Shooter extends SimEntityBase{

// instance variables
   private Coordinate shooterLocation;
   private double shooterRange;
   private double probableErrorRange;
   private double probableErrorDeflection;
   private double maxRateOfFire;
   private String acquireServiceDistribution;
   private String firingServiceDistribution;
   private RandomVariate acquireServiceTime;
   private RandomVariate firingServiceTime;
   private RandomVariate shotRangeError;
   private RandomVariate shotDeflError;
   private int numberRounds;
   private int startNumberRounds;
   private int burstRadius;
   private double DestroyBurstRadius;
   private double NeutralizeBurstRadius;
   private double SuppressBurstRadius;
   private double meanProjectileVelocity;
   private String platformType;
   private int numberGuns;

// constructor methods
   public Shooter(Coordinate theShooterLocation, double theShooterRange,
     double thePER, double thePED, String theAcquireServiceDistribution,
     Object[] theAcquireParameters, String theFiringServiceDistribution,
     Object[] theFiringParameters, double theMaxRateOfFire, long firingSeed,
     long acquireSeed, int rounds, int theBurstRadius,double theMeanProjectileVelocity,
     String thePlatformType, int theNumberGuns){
        acquireServiceTime = RandomFactory.getRandomVariate(theAcquireServiceDistribution,
         theAcquireParameters,acquireSeed);
        firingServiceTime = RandomFactory.getRandomVariate(theFiringServiceDistribution,
         theFiringParameters,firingSeed);
      shotRangeError = RandomFactory.getRandomVariate("Normal",
        new Object[]{new Double(0.0),new Double(thePER)});
      shotDeflError = RandomFactory.getRandomVariate("Normal",new Object[]{new
Double(0.0),
        new Double(thePED)});
        setAcquireServiceDistribution(theAcquireServiceDistribution);
```

```
         setFiringServiceDistribution(theFiringServiceDistribution);
       shooterLocation = new Coordinate(theShooterLocation);
       shooterRange = theShooterRange;
       probableErrorRange = thePER;
       probableErrorDeflection = thePED;
       startNumberRounds = rounds;
       numberRounds = rounds;
       burstRadius = theBurstRadius;
       DestroyBurstRadius = Math.sqrt(18*burstRadius*burstRadius);
       NeutralizeBurstRadius = Math.sqrt(6*burstRadius*burstRadius);
       SuppressBurstRadius = 2*Math.sqrt(3*burstRadius*burstRadius);
       meanProjectileVelocity = theMeanProjectileVelocity;
       platformType = new String(thePlatformType);
       maxRateOfFire = theMaxRateOfFire;
       numberGuns = theNumberGuns;
    }

 // class methods

    public boolean inRange(Target theTarget){
       return (shooterLocation.distanceFrom(theTarget.getTargetLocation())<=shooterRange);
    }

    public Coordinate getShooterLocation(){return new Coordinate(shooterLocation);}

    public double getShooterRange(){return shooterRange;}

    public void setShooterRange(double theShooterRange){
       shooterRange = theShooterRange;
    }

    public double getPER(){return probableErrorRange;}

    public void setPER(double thePER){
       probableErrorRange = thePER;
    }

    public double getPED(){return probableErrorDeflection;}

    public void setPED(double thePED){
       probableErrorDeflection = thePED;
    }

    public void setAcquireServiceDistribution(String newDistribution){
         acquireServiceDistribution = newDistribution;
    }

    public void setFiringServiceDistribution(String newDistribution){
         firingServiceDistribution = newDistribution;
    }

    public String getAcquireDistribution(){return
 acquireServiceTime.getClass().getName();}

    public String getFiringDistribution(){return firingServiceTime.getClass().getName();}

    public void setAcquireParameters(Object[] newParameters){
         acquireServiceTime.setParameters(newParameters);
    }

    public void setFiringParameters(Object[] newParameters){
         firingServiceTime.setParameters(newParameters);
    }

    public Object[] getAcquireParameters(){return acquireServiceTime.getParameters();}

    public Object[] getFiringParameters(){return firingServiceTime.getParameters();}

    public double getServiceTime(Target theTarget){
       int volleys = (int)Math.ceil((double)theTarget.getSalvoSize()/numberGuns);;
       double tgtRange = shooterLocation.distanceFrom(theTarget.getTargetLocation());
```

```
      double tof = tgtRange/meanProjectileVelocity;
      double firingTime = firingServiceTime.generate();
      double acquireTime = acquireServiceTime.generate();
      if (firingTime<0.0){
         firingTime = 0.0 + volleys/maxRateOfFire;
      }
      else{
         firingTime += volleys/maxRateOfFire;;
      }
      if (acquireTime<0.0){
         acquireTime = 0.0;
      }
      return firingTime + acquireTime + tof;
   }

   public double getRepeatServiceTime(Target theTarget){
      int volleys = (int)Math.ceil((double)theTarget.getSalvoSize()/numberGuns);;
      double firingTime = volleys/maxRateOfFire;
      double tgtRange = shooterLocation.distanceFrom(theTarget.getTargetLocation());
      double tof = tgtRange/meanProjectileVelocity;
      return firingTime + tof;
   }

   public int getNumberRounds(){return numberRounds;}

   public boolean hasRounds(){return (numberRounds>0);}

   public void decrementRounds(int roundsFired){
      numberRounds -= roundsFired;
   }

   public void incrementRounds(int resupply){
      numberRounds+=resupply;
   }

   public Coordinate getSingleShotLocation(Coordinate theTarget){
      double tgtRange = shooterLocation.distanceFrom(theTarget);
      double xVec = (theTarget).getXCoord()-shooterLocation.getXCoord();
      double angle = Math.acos(xVec/tgtRange);
      double deflectionError = tgtRange/1000*shotDeflError.generate();
      double rangeError = shotRangeError.generate();
      double deltaX = Math.cos(angle)*rangeError + Math.sin(angle)*deflectionError;
      double deltaY = Math.sin(angle)*rangeError - Math.cos(angle)*deflectionError;
      Coordinate shotLocation = (theTarget).incrementBy(new Coordinate(deltaX,deltaY));
      return new Coordinate(shotLocation);
   }

   public int getBurstRadius(){return burstRadius;}

   public String getPlatformType(){return new String(platformType);}

   public void reset(){
      super.reset();
      numberRounds = startNumberRounds;
   }

   public String toString(){
           return "Shooter is: " + platformType + ", its location is: " +
        this.getShooterLocation() + ", shooter range is: " + this.getShooterRange() +
        ", shooter PER is: " + this.getPER() +
        ", shooter PED is: " + this.getPED() +
        ", shooter has: " + this.getNumberRounds() + " rounds remaining.";
   }

}
```

4. Target

```
/**
 * Juan K. Ulloa <BR>
 * Thesis: FSST
 * January 13, 2001
 *
 * <P> This class creates an instance of a target that stores the
 *     time that the call for fire (CFF) is initially made.
 *     This time is the arrival time of the target.
**/

package fsst;
import simkit.*;
import simkit.data.*;
import simkit.smd.*;
import java.util.*;

public class Target{

// instance variables
   private BattleFieldData battleField;
   private TargetState state;
   private double arrivalTime;
   private Coordinate targetLocation;
   private String targetType;
   private String mission;
   private int salvoSize;
   private int closeCalls;
   private int hits;
   private int hitsNeeded;
   private int hitsToDestroy;
   private int hitsToNeutralize;
   private int closeCallsToSuppress;
   private double collateralDamageRadius;
   private double collateralDamagePercentage;
   private Random r;

// constructor methods
   public Target(){
        arrivalTime = Schedule.getSimTime();
   }

   public Target(AO theBox, BattleFieldData theBattleField){
      state = TargetState.UNSCATHED;
        arrivalTime = Schedule.getSimTime();
      targetLocation = new Coordinate(theBox.getRandomLocation());
      battleField = theBattleField;
      targetType = battleField.getTargetType();
      mission = battleField.getFireMissionType();
      salvoSize = battleField.getSalvoSize(targetType,mission);
      hitsToDestroy = battleField.getDestroy(targetType);
      hitsToNeutralize = battleField.getNeutralize(targetType);
      closeCallsToSuppress = battleField.getSuppress(targetType);
      hitsNeeded = battleField.getHitsNeeded(targetType,mission);
      collateralDamageRadius = battleField.getCollateralDamageRadius();
      collateralDamagePercentage = battleField.getCollateralDamagePercent();
   }

// class methods
   public double getArrivalTime(){return arrivalTime;}

   public String getTargetState(){return state.toString();}

   public Coordinate getTargetLocation(){return new Coordinate(targetLocation);}

   public String getTargetType(){return new String(targetType);}

   public String getMission(){return new String(mission);}

   public int getSalvoSize(){return salvoSize;}

   public int getHitsNeeded(){return hitsNeeded;}

   public int getHits(){return hits;}
   public int getCloseCalls(){return closeCalls;}

   public boolean isFireMissionSuccessful(Shooter theShooter){
      int burstRadius = theShooter.getBurstRadius();
      double missedBy;
      for(int i=0;i<salvoSize;i++){
         missedBy = (theShooter.getSingleShotLocation(new Coordinate(targetLocation))).
           distanceFrom(targetLocation);
         if (missedBy <= burstRadius){
            hits++;
         }
         if (missedBy > battleField.getCollateralDamageRadius()){
            if (battleField.getRandomNumber()<battleField.getCollateralDamagePercent()){
               battleField.increaseNumberCollateralDamageCausingRounds();
            }
         }
         if ((missedBy <= (2*burstRadius))&&(mission.equals("Suppress"))){
            closeCalls++;
         }
      }
      setState();
      if (mission.equals("Suppress")){
         return (closeCalls>=hitsNeeded);
      }
      else {
         return (hits>=hitsNeeded);
      }
   }

   public void setState(){
      if (closeCalls>=closeCallsToSuppress){
         state = TargetState.SUPPRESSED;
      }
      if (hits>=hitsToNeutralize){
         state = TargetState.NEUTRALIZED;
      }
      if (hits>=hitsToDestroy){
         state = TargetState.DESTROYED;
      }
   }

   public void setSalvoSize(int newSalvoSize){
      salvoSize = newSalvoSize;
   }

   public String toString(){
          return getTargetType() + " arrived at " + getArrivalTime() + ",at location: " +
        this.getTargetLocation() + ".  The mission was to: " + this.getMission() +
        ".  The target is currently " + state.toString();
   }

}
```

5. TargetState

```
/**
 * Juan K. Ulloa <BR>
 * Thesis: FSST
 * January 13, 2001
 *
 * <P> This class tracks the state of the target being engaged.
 *
**/

package fsst;

public class TargetState{

   public static final TargetState UNSCATHED= new TargetState("Unscathed");
   public static final TargetState SUPPRESSED = new TargetState("Suppressed");
   public static final TargetState NEUTRALIZED = new TargetState("Neutralized");
   public static final TargetState DESTROYED = new TargetState("Destroyed");

   private String state;

   protected TargetState(String theState){
      state = new String(theState);
   }

   public String toString(){
      return state;
   }

}
```

6. TargetArrivalProcess

```
/**
 * Juan K. Ulloa <BR>
 * Thesis: FSST
 * January 12, 2001
 *
 * <P> This class simulates the arrival process of targets.
 *
**/

package fsst;
import simkit.*;
import simkit.data.*;

public class TgtArrivalProcess extends SimEntityBase {

// instance variables
   private int numberArrivals;
   private RandomVariate arrivalTimeGenerator;

// constructor methods
   public TgtArrivalProcess(String distribution, Object[] theParameters, long seed){
      arrivalTimeGenerator =
        RandomFactory.getRandomVariate(distribution,theParameters,seed);
   }

// instance methods
   public String getDistribution(){return arrivalTimeGenerator.getClass().getName();}

   public void setParameters(Object[] newParameters){
      arrivalTimeGenerator.setParameters(newParameters);
   }

   public Object[] getParameters(){return arrivalTimeGenerator.getParameters();}

   public int getNumberArrivals(){return numberArrivals;}

   public void doRun(){
      double temp = arrivalTimeGenerator.generate();
      if (temp<0.0){
         temp = 0.0;
      }
      numberArrivals = 0;
      waitDelay("TargetArrival",temp);
   }

   public void doTargetArrival(){
      double temp = arrivalTimeGenerator.generate();
      if (temp<0.0){
         temp = 0.0;
      }
      numberArrivals++;
      waitDelay("TargetArrival",temp);
   }

   public void reset(){
      super.reset();
      numberArrivals = 0;
   }

}
```

7. TargetServer2

```
/**
 * Juan K. Ulloa <BR>
 * Thesis: FSST
 * January 19, 2001
 *
 * <P> This class implements the process of firing targets.
 *
**/

package fsst;
import simkit.*;
import simkit.data.*;
import java.util.*;

public class TgtServer2 extends SimEntityBase{

// instance variables
        // parameters
   private int maxNumberShooters;
        // state variables
   private Vector queue = new Vector();
   private int queueSize;
   private Vector shooters;
   private int numberMissionRepeats;
   private int numberAvailableShooters;
   private int numberFireMissionsServed;
   private int numberMissionsRejected;
   private int numberSuccessfulMissions;
   private int numberUnsuccessfulMissions;
   private AO box;
   private BattleFieldData battleFieldData;
   private double[] weights;

// constructor methods
   public TgtServer2(Vector theShooters, AO theBox, BattleFieldData theBattleFieldData,
     double[] theWeights){
      battleFieldData = new BattleFieldData(theBattleFieldData);
      box = new AO(theBox);
        setNumberShooters(theShooters.size());
      shooters = (Vector)theShooters.clone();
      numberAvailableShooters = maxNumberShooters;
      for(int i=0;i<shooters.size();i++){
         queue.add(new LinkedList());
      }
      weights = theWeights;
   }

// instance methods
   public void doTargetArrival(){
        Target newFireMission = new Target(box,battleFieldData);
      Vector whoIsInRange = new Vector();
      int firer;
      for(int i=0;i<shooters.size();i++){
         if (((Shooter)shooters.get(i)).inRange(newFireMission)){
            whoIsInRange.add(new Integer(i));
         }
      }
      if (whoIsInRange.size()>0) {
         firer = getShooter1(whoIsInRange,newFireMission);
         if (firer>=0){
            ((LinkedList)queue.get(firer)).add(newFireMission);
               firePropertyChange("tgtsInQueue",queueSize,++queueSize);
          if (((LinkedList)queue.get(firer)).size() == 1){
               waitDelay("FireMission",0.0,new Integer(firer));
            }
         }
         else{
            firePropertyChange("numberMissionsRejected",numberMissionsRejected,
              ++numberMissionsRejected);
         }
      }
      else {
         firePropertyChange("numberMissionsRejected",numberMissionsRejected,
           ++numberMissionsRejected);
      }
   }

   public void doFireMission(Integer theFirer){
      int firer = theFirer.intValue();
       Target thisFireMission = (Target)((LinkedList)queue.get(firer)).getFirst();
      double thisTimeInQueue = Schedule.getSimTime() - thisFireMission.getArrivalTime();
      Vector parameters = new Vector();
      parameters.add(thisFireMission);
      parameters.add(theFirer);
       firePropertyChange("tgtsInQueue",queueSize,--queueSize);
       firePropertyChange("numberAvailableShooters",numberAvailableShooters,
        --numberAvailableShooters);
      if((((Shooter)shooters.get(firer)).getNumberRounds()<
        thisFireMission.getSalvoSize())&&(((Shooter)shooters.get(firer)).hasRounds())){
         thisFireMission.setSalvoSize(((Shooter)shooters.get(firer)).getNumberRounds());
      }
// Check to see if shooter has enough rounds left to fire
      if (!((Shooter)shooters.get(firer)).hasRounds()){
          waitDelay("FireMissionNeverFired",0.0,parameters);
      }
   // mission successful
      else if (thisFireMission.isFireMissionSuccessful((Shooter)shooters.get(firer))) {
         ((Shooter)shooters.get(firer)).decrementRounds(thisFireMission.getSalvoSize());
               firePropertyChange("fireMissionQueueTime",null,new
Double(thisTimeInQueue));
         parameters.add(new Boolean(true));
          waitDelay("EndFireMission",((Shooter)shooters.get(firer)).
           getServiceTime(thisFireMission),parameters);
      }
   // mission unsuccessful and either the target flees or shooter out of ammo
      else if ((battleFieldData.didFlee())||
        (!((Shooter)shooters.get(firer)).hasRounds())){
         ((Shooter)shooters.get(firer)).decrementRounds(thisFireMission.getSalvoSize());
               firePropertyChange("fireMissionQueueTime",null,new
Double(thisTimeInQueue));
         parameters.add(new Boolean(false));
          waitDelay("EndFireMission",((Shooter)shooters.get(firer)).
           getServiceTime(thisFireMission),parameters);
      }
   // mission unsucccessful, but target still available and shooter has ammo
      else {
         firePropertyChange("numberMissionRepeats",numberMissionRepeats,
           ++numberMissionRepeats);
         waitDelay("RepeatFireMission",((Shooter)shooters.get(firer)).
           getServiceTime(thisFireMission),parameters);
      }
   }

   public void doRepeatFireMission(Vector theParameters){
      Target thisFireMission = (Target)theParameters.get(0);
      int firer = ((Integer)theParameters.get(1)).intValue();
      double thisTimeInQueue = Schedule.getSimTime() - thisFireMission.getArrivalTime();
      if((((Shooter)shooters.get(firer)).getNumberRounds()<
        thisFireMission.getSalvoSize())&&(((Shooter)shooters.get(firer)).hasRounds())){
         thisFireMission.setSalvoSize(((Shooter)shooters.get(firer)).getNumberRounds());
      }
// Check to see if shooter has enough rounds left to fire the mission
      if (!((Shooter)shooters.get(firer)).hasRounds()){
        firePropertyChange("fireMissionQueueTime",null,new Double(thisTimeInQueue));
```

```
      theParameters.add(new Boolean(false));
        waitDelay("EndFireMission",0.0,theParameters);
      }
// mission successful
   else if (thisFireMission.isFireMissionSuccessful((Shooter)shooters.get(firer))) {
      ((Shooter)shooters.get(firer)).decrementRounds(thisFireMission.getSalvoSize());
     firePropertyChange("fireMissionQueueTime",null,new Double(thisTimeInQueue));
      theParameters.add(new Boolean(true));
        waitDelay("EndFireMission",((Shooter)shooters.get(firer)).
         getRepeatServiceTime(thisFireMission),theParameters);
   }
// mission unsuccessful and (target flees or shooter out of ammo)
   else if ((battleFieldData.didFlee())||
     (!((Shooter)shooters.get(firer)).hasRounds())){
      ((Shooter)shooters.get(firer)).decrementRounds(thisFireMission.getSalvoSize());
     firePropertyChange("fireMissionQueueTime",null,new Double(thisTimeInQueue));
      theParameters.add(new Boolean(false));
        waitDelay("EndFireMission",((Shooter)shooters.get(firer)).
         getRepeatServiceTime(thisFireMission),theParameters);
   }
// mission unsuccessful and target still available
   else{
      ((Shooter)shooters.get(firer)).decrementRounds(thisFireMission.getSalvoSize());
      waitDelay("RepeatFireMission",((Shooter)shooters.get(firer)).
        getRepeatServiceTime(thisFireMission),theParameters);
   }
}

public void doEndFireMission(Vector theParameters){
   Target theFireMission = (Target)theParameters.get(0);
   int firer = ((Integer)theParameters.get(1)).intValue();
   boolean success = ((Boolean)theParameters.get(2)).booleanValue();
     double timeInSystem = Schedule.getSimTime()-theFireMission.getArrivalTime();
   ((LinkedList)queue.get(firer)).removeFirst();
   if (success){
      firePropertyChange("numberSuccessfulMissions",numberSuccessfulMissions,
        ++numberSuccessfulMissions);
   }
   else{
      firePropertyChange("numberUnsuccessfulMissions",numberUnsuccessfulMissions,
        ++numberUnsuccessfulMissions);
   }
     firePropertyChange("totalFireMissionTime",null,new Double(timeInSystem));
     firePropertyChange("numberMissionsFired",numberFireMissionsServed,
      ++numberFireMissionsServed);
     firePropertyChange("numberAvailableShooters",numberAvailableShooters,
      ++numberAvailableShooters);
     if (((LinkedList)queue.get(firer)).size() > 0){
           waitDelay("FireMission",0.0,new Integer(firer));
     }
}

public void doFireMissionNeverFired(Vector theParameters){
   Target theFireMission = (Target)theParameters.get(0);
   int firer = ((Integer)theParameters.get(1)).intValue();
   for(int i=0;i<((LinkedList)queue.get(firer)).size();i++){
      firePropertyChange("numberMissionsRejected",numberMissionsRejected,
        ++numberMissionsRejected);
   }
   ((LinkedList)queue.get(firer)).clear();
}

public int getNumberShooters(){return maxNumberShooters;}

public void setNumberShooters(int numberShooters){
     maxNumberShooters = numberShooters;
}

public int getNumberServed(){return numberFireMissionsServed;}

public int getNumberInQueue(){return queueSize;}
public void addShooter(Shooter newShooter){
   shooters.add(newShooter);
   queue.add(new LinkedList());
    firePropertyChange("numberAvailableShooters",numberAvailableShooters,
     ++numberAvailableShooters);
   maxNumberShooters++;
}

public void removeShooter(int shooterNumber){
   int numberMissions = ((LinkedList)queue.get(shooterNumber)).size();
   for(int i=0;i<numberMissions;i++){
      // add the old targets
   }
   shooters.remove(shooterNumber);
   queue.remove(shooterNumber);
    firePropertyChange("numberAvailableShooters",numberAvailableShooters,
     --numberAvailableShooters);
   maxNumberShooters--;
}

public int getShooter(Vector inRangeShooters, Target thisFireMission){
   int firer = 0;
   int temp;
   Shooter thisShooter;
   Shooter bestShooterSoFar;
   double thisShooterDistance;
   double bestShooterSoFarDistance;
   Vector inRangeFA = new Vector();
   Vector inRangeNSFS = new Vector();
// create vector of inRange shooters that have enough ammo and are FA
   for(int i=0;i<inRangeShooters.size();i++){
      temp = ((Integer)inRangeShooters.get(i)).intValue();
      thisShooter = (Shooter)shooters.get(temp);
      if (thisShooter.getPlatformType().equals("FieldArtillery")&&
        (thisShooter.getNumberRounds()>=thisFireMission.getSalvoSize())){
         inRangeFA.add(new Integer(temp));
      }
      if(thisShooter.getPlatformType().equals("NSFS")&&
        (thisShooter.getNumberRounds()>=thisFireMission.getSalvoSize())){
         inRangeNSFS.add(new Integer(temp));
      }
   }
      // Use the FA assets first
   if (inRangeFA.size()>0){
      firer = ((Integer)inRangeFA.get(0)).intValue();
      for(int i=0;i<inRangeFA.size();i++){
         temp = ((Integer)inRangeFA.get(i)).intValue();
         thisShooter = (Shooter)shooters.get(temp);
         thisShooterDistance = (thisShooter.getShooterLocation()).
           distanceFrom(thisFireMission.getTargetLocation());
         bestShooterSoFar = (Shooter)shooters.get(firer);
         bestShooterSoFarDistance = (bestShooterSoFar.getShooterLocation()).
           distanceFrom(thisFireMission.getTargetLocation());
// use asset enough rounds and with shortest queue
         if ((((LinkedList)queue.get(temp)).size()<=
           ((LinkedList)queue.get(firer)).size())){
            firer = temp;
         }
// use asset with lowest PER if all else is the same
         else if((((LinkedList)queue.get(temp)).size()==
           ((LinkedList)queue.get(firer)).size())&&
           (thisShooter.getPER()<bestShooterSoFar.getPER())){
            firer = temp;
         }
// use asset that is closest if all else is the same
         else if((((LinkedList)queue.get(temp)).size()==
           ((LinkedList)queue.get(firer)).size())&&
           (thisShooter.getPER()<bestShooterSoFar.getPER())&&
           (thisShooterDistance<bestShooterSoFarDistance)){
            firer = temp;
         }
      }
      return firer;
   }
      // Use the NSFS assets if no FA assets available
   if (inRangeNSFS.size()>0){
      firer = ((Integer)inRangeNSFS.get(0)).intValue();
      for(int i=0;i<inRangeNSFS.size();i++){
         temp = ((Integer)inRangeNSFS.get(i)).intValue();
         thisShooter = (Shooter)shooters.get(temp);
         thisShooterDistance = (thisShooter.getShooterLocation()).
           distanceFrom(thisFireMission.getTargetLocation());
         bestShooterSoFar = (Shooter)shooters.get(firer);
         bestShooterSoFarDistance = (bestShooterSoFar.getShooterLocation()).
           distanceFrom(thisFireMission.getTargetLocation());
// use asset enough rounds and with shortest queue
         if ((((LinkedList)queue.get(temp)).size()<=
           ((LinkedList)queue.get(firer)).size())){
            firer = temp;
         }
// use asset with lowest PER if all else is the same
         else if((((LinkedList)queue.get(temp)).size()==
           ((LinkedList)queue.get(firer)).size())&&
           (thisShooter.getPER()<bestShooterSoFar.getPER())){
            firer = temp;
         }
// use asset that is closest if all else is the same
         else if((((LinkedList)queue.get(temp)).size()==
           ((LinkedList)queue.get(firer)).size())&&
           (thisShooter.getPER()<bestShooterSoFar.getPER())&&
           (thisShooterDistance<bestShooterSoFarDistance)){
            firer = temp;
         }
      }
      return firer;
   }
// if no firers have enough ammo, use asset with the most ammo and reset salvoSize
// to number of rounds available
   firer = -1;
   for(int i=0;i<inRangeShooters.size();i++){
      temp = ((Integer)inRangeShooters.get(i)).intValue();
      thisShooter = (Shooter)shooters.get(temp);
      if (firer>=0){
         bestShooterSoFar = (Shooter)shooters.get(firer);
      }
      else{
         bestShooterSoFar = (Shooter)shooters.get(temp);
      }
      if ((thisShooter.hasRounds())&&
        (thisShooter.getNumberRounds()>bestShooterSoFar.getNumberRounds())){
         firer = temp;
         thisFireMission.setSalvoSize(thisShooter.getNumberRounds());
      }
   }
   return firer;
}

public int getShooter1(Vector inRangeShooters, Target thisFireMission){
   Shooter thisShooter;
   Shooter bestShooterSoFar;
   Vector inRangeWithEnoughAmmo = new Vector();
   Vector inRangeNotEnoughAmmo = new Vector();
   Vector weightedShooters;
   double[] theValues;
   double shooterWorth;
   double bestShooter = 0.0;
   int temp;
   int firer = -1;
// create vector of inRange shooters that have enough ammo
   for(int i=0;i<inRangeShooters.size();i++){
      temp = ((Integer)inRangeShooters.get(i)).intValue();
```

```
         thisShooter = (Shooter)shooters.get(temp);
         if (thisShooter.getNumberRounds()>=thisFireMission.getSalvoSize()){
            inRangeWithEnoughAmmo.add(new Integer(temp));
         }
         else{
            inRangeNotEnoughAmmo.add(new Integer(temp));
         }
      }
      if (inRangeWithEnoughAmmo.size()>0){
         weightedShooters = new Vector();
         theValues = getShooterBestWorstValues(inRangeWithEnoughAmmo,thisFireMission);
//{bestRange,worstRange,bestPER,worstPER,bestNumberRounds,worstNumberRounds};
         for(int i=0;i<inRangeWithEnoughAmmo.size();i++){
            temp = ((Integer)inRangeWithEnoughAmmo.get(i)).intValue();
            thisShooter = (Shooter)shooters.get(temp);
            double range = thisShooter.getShooterLocation().
              distanceFrom(thisFireMission.getTargetLocation());
            double per = thisShooter.getPER();
            int numberRounds = thisShooter.getNumberRounds();
      // shooter worth is function of range, PER, number rounds, and platform type
            shooterWorth = (range-theValues[1])/(theValues[0]-theValues[1])*weights[0]+
              (per-theValues[3])/(theValues[2]-theValues[3])*weights[1]+
              (numberRounds-theValues[5])/(theValues[4]-theValues[5])*weights[2];
            if (thisShooter.getPlatformType().equals("FieldArtillery")){
               shooterWorth += weights[3];
            }
            if (thisShooter.getPlatformType().equals("NSFS")){
               shooterWorth += weights[4];
            }
            weightedShooters.add(new Double(shooterWorth));
         }
         for(int i=0;i<weightedShooters.size();i++){
            shooterWorth = ((Double)weightedShooters.get(i)).doubleValue();
//            System.out.println(shooterWorth);
            if (shooterWorth>bestShooter){
               firer = ((Integer)inRangeWithEnoughAmmo.get(i)).intValue();
               bestShooter = shooterWorth;
            }
         }
         return firer;
      }
      else{
         for(int i=0;i<inRangeNotEnoughAmmo.size();i++){
            temp = ((Integer)inRangeShooters.get(i)).intValue();
            thisShooter = (Shooter)shooters.get(temp);
            if (firer>=0){
               bestShooterSoFar = (Shooter)shooters.get(firer);
            }
            else{
               bestShooterSoFar = (Shooter)shooters.get(temp);
            }
            if(thisShooter.getNumberRounds()>bestShooterSoFar.getNumberRounds()){
               firer = temp;
               thisFireMission.setSalvoSize(thisShooter.getNumberRounds());
            }
         }
         return firer;
      }
   }

   public int getNumberRoundsCausingCollateralDamage(){
      return battleFieldData.getNumberCollateralDamageCausingRounds();
   }

   public double[] getShooterBestWorstValues(Vector theShooters, Target theTarget){
      int temp;
      double bestRange = Double.MAX_VALUE;
      double worstRange = 0.0;
      double bestPER = Double.MAX_VALUE;
      double worstPER = 0.0;
      int bestNumberRounds = 0;
```

```
        int worstNumberRounds = Integer.MAX_VALUE;
        Shooter currentShooter;
        for(int i=0;i<theShooters.size();i++){
           temp = ((Integer)theShooters.get(i)).intValue();
           currentShooter = (Shooter)shooters.get(temp);
           if ((currentShooter.getShooterLocation()).
             distanceFrom(theTarget.getTargetLocation()) > worstRange){
              worstRange = (currentShooter.getShooterLocation()).
                distanceFrom(theTarget.getTargetLocation());
           }
           if ((currentShooter.getShooterLocation()).
             distanceFrom(theTarget.getTargetLocation()) < bestRange){
              bestRange = (currentShooter.getShooterLocation()).
                distanceFrom(theTarget.getTargetLocation());
           }
           if (currentShooter.getPER()>worstPER){
              worstPER = currentShooter.getPER();
           }
           if (currentShooter.getPER()<bestPER){
              bestPER = currentShooter.getPER();
           }
           if (currentShooter.getNumberRounds()>bestNumberRounds){
              bestNumberRounds = currentShooter.getNumberRounds();
           }
           if (currentShooter.getNumberRounds()<worstNumberRounds){
              worstNumberRounds = currentShooter.getNumberRounds();
           }
        }
        if (worstRange==bestRange){
           worstRange+=0.1;
        }
        if (worstPER==bestPER){
           worstPER+=0.1;
        }
        if (worstNumberRounds==bestNumberRounds){
           worstNumberRounds-=0.1;
        }
        return new double[]{bestRange,worstRange,bestPER,
          worstPER,bestNumberRounds,worstNumberRounds};
    }

    public void reset(){
          super.reset();
        numberMissionRepeats = 0;
        firePropertyChange("numberMissionRepeats",Integer.MIN_VALUE,numberMissionRepeats);
          numberFireMissionsServed = 0;

          firePropertyChange("numberMissionsFired",Integer.MIN_VALUE,numberFireMissionsServe
d);
        numberUnsuccessfulMissions = 0;
        firePropertyChange("numberUnsuccessfulMissions",Integer.MIN_VALUE,
          numberUnsuccessfulMissions);
        numberSuccessfulMissions = 0;
        firePropertyChange("numberSuccessfulMissions",Integer.MIN_VALUE,
          numberSuccessfulMissions);
        numberMissionsRejected = 0;
        firePropertyChange("numberMissionsRejected",Integer.MIN_VALUE,
          numberMissionsRejected);
          numberAvailableShooters = maxNumberShooters;
          firePropertyChange("numberAvailableShooters",Integer.MIN_VALUE,
          numberAvailableShooters);
          queue.clear();
        for(int i=0;i<shooters.size();i++){
           queue.add(new LinkedList());
        }
          firePropertyChange("tgtsInQueue",Integer.MIN_VALUE,queue.size());
          firePropertyChange("fireMissionQueueTime",null,new Double(0.0));
          firePropertyChange("totalFireMissionTime",null,new Double(0.0));
        battleFieldData.resetCollateralDamageCausingRounds();
    }
}
```

B. MAIN METHODS

1. FSST50Replications.java

```
/**
 * Juan K. Ulloa <BR>
 * Thesis: FSST
 * January 19, 2001
 *
 * <P>
 *
**/

package fsst;
import simkit.*;
import simkit.data.*;
import simkit.smd.*;
import java.util.*;
import java.io.*;
import java.text.DecimalFormat;

public class FSST50Replications{

// main method
   public static final int NUMREPLICATIONS = 50;
   public static void main(String[] args){
      DecimalFormat deci = new DecimalFormat(" 0.0000;-0.0000");
        Properties props = new Properties();
           try {
                props.load(new FileInputStream(args[0]));
           }
           catch (FileNotFoundException e) {System.err.println(e);}
           catch (IOException e) {System.err.println(e);}

          // get COA data and CA
        String theCOA = props.get("COA").toString();
       double ca = Double.parseDouble(props.get("theCA").toString());
          // get AO data
        double xLowerLeft = Double.parseDouble(props.get("xLowerLeft").toString());
        double yLowerLeft = Double.parseDouble(props.get("yLowerLeft").toString());
        double xUpperRight = Double.parseDouble(props.get("xUpperRight").toString());
        double yUpperRight= Double.parseDouble(props.get("yUpperRight").toString());
        String xDist = props.get("xDist").toString();
        double xDistA = Double.parseDouble(props.get("xDistA").toString());
        double xDistB = Double.parseDouble(props.get("xDistB").toString());
        String yDist = props.get("yDist").toString();
        double yDistMean = Double.parseDouble(props.get("yDistMean").toString());

          // instantiate AO
      Coordinate lowerLeft = new Coordinate(xLowerLeft,yLowerLeft);
      Coordinate upperRight = new Coordinate(xUpperRight,yUpperRight);
      Object[] xDistParameters = new Object[]{new Double(xDistA),new Double(xDistB)};
      Object[] yDistParameters = new Object[]{new Double(yDistMean)};
      AO box = new AO(lowerLeft,upperRight,xDist,xDistParameters,yDist,yDistParameters);

        // get the stop data
        String stopEvent = props.get("stopEvent").toString();
        int stopEventCount = Integer.parseInt(props.get("stopEventCount").toString());

        // get the ArrivalProcess data
        String interarrivalDistribution = props.get("arrivalDistribution").toString();
        double meanInterarrivalTime =
          Double.parseDouble(props.get("meanInterarrival").toString());
        int arrivalStream = Integer.parseInt(props.get("arrivalStream").toString());

        // instantiate the arrivalProcess
        TgtArrivalProcess arrival = new TgtArrivalProcess(interarrivalDistribution,
          new Object[] {new 
Double(meanInterarrivalTime)},RandomStream.STREAM[arrivalStream]);

        // get the Shooter data
      int numberShooters = Integer.parseInt(props.get("numberShooters").toString());
        double meanProjectileVelocity =
         Double.parseDouble(props.get("meanProjectileVelocity").toString());
      String[] platformType = new String[numberShooters];
      String[] acquireServiceDistribution = new String[numberShooters];
      double[] meanAcquireInterservice = new double[numberShooters];
      double[] sigmaAcquire = new double[numberShooters];
      String[] firingServiceDistribution = new String[numberShooters];
      double[] meanFiringInterservice = new double[numberShooters];
      double[] sigmaFiring = new double[numberShooters];
      double[] maxRateOfFire = new double[numberShooters];
      double[] thePER = new double[numberShooters];
      double[] thePED = new double[numberShooters];
      double[] shooterXCoord = new double[numberShooters];
      double[] shooterYCoord = new double[numberShooters];
      int[] acquireServerStream = new int[numberShooters];
      int[] firingServerStream = new int[numberShooters];
      int[] numberRounds = new int[numberShooters];
      int[] numberGuns = new int[numberShooters];
      int[] shooterRange = new int[numberShooters];
      int[] burstRadius = new int[numberShooters];
      Shooter[] aShooter = new Shooter[numberShooters];

      for(int i=0;i<numberShooters;i++){
         platformType[i] = props.get("platformType"+(i+1)).toString();
        acquireServiceDistribution[i] =
           props.get("acquireServiceDistribution"+(i+1)).toString();
          meanAcquireInterservice[i] =
           Double.parseDouble(props.get("meanAcquireInterservice"+(i+1)).toString());
        sigmaAcquire[i] = Double.parseDouble(props.get("sigmaAcquire"+(i+1)).toString());
        firingServiceDistribution[i] =
           props.get("firingServiceDistribution"+(i+1)).toString();
          meanFiringInterservice[i] =
           Double.parseDouble(props.get("meanFiringInterservice"+(i+1)).toString());
        sigmaFiring[i] = Double.parseDouble(props.get("sigmaFiring"+(i+1)).toString());
         maxRateOfFire[i] =
           Double.parseDouble(props.get("maxRateOfFire"+(i+1)).toString());
        thePER[i] = Double.parseDouble(props.get("thePER"+(i+1)).toString());
         thePED[i] = Double.parseDouble(props.get("thePED"+(i+1)).toString());
         shooterXCoord[i] =
Double.parseDouble(props.get("shooterXCoord"+(i+1)).toString());
         shooterYCoord[i] =
Double.parseDouble(props.get("shooterYCoord"+(i+1)).toString());
        acquireServerStream[i] =
           Integer.parseInt(props.get("acquireServerStream"+(i+1)).toString());
        firingServerStream[i] =
           Integer.parseInt(props.get("firingServerStream"+(i+1)).toString());
        numberRounds[i] = Integer.parseInt(props.get("numberRounds"+(i+1)).toString());
        numberGuns[i] = Integer.parseInt(props.get("numberGuns"+(i+1)).toString());
         shooterRange[i] = Integer.parseInt(props.get("shooterRange"+(i+1)).toString());
         burstRadius[i] = Integer.parseInt(props.get("burstRadius"+(i+1)).toString());
            // instantiate the shooters
         aShooter[i] = new Shooter(new Coordinate(shooterXCoord[i],shooterYCoord[i]),
           shooterRange[i],thePER[i],thePED[i],acquireServiceDistribution[i],
           new Object[] {new Double(meanAcquireInterservice[i]),
           new Double(sigmaAcquire[i])},firingServiceDistribution[i],new Object[]
           {new Double(meanFiringInterservice[i]),new Double(sigmaFiring[i])},
           maxRateOfFire[i],RandomStream.STREAM[firingServerStream[i]],
           RandomStream.STREAM[acquireServerStream[i]],numberRounds[i],
           burstRadius[i],meanProjectileVelocity, platformType[i],numberGuns[i]);
      }

         // instantiate the BattleFieldData
      double fleeProbability =
Double.parseDouble(props.get("fleeProbability").toString());
      int fleeStream = Integer.parseInt(props.get("fleeStream").toString());
      double collateralDamageRadius =
        Double.parseDouble(props.get("collateralDamageRadius").toString());
      double collateralDamagePercent =
        Double.parseDouble(props.get("collateralDamagePercent").toString());
      double percentArmor =
        Double.parseDouble(props.get("percentArmor").toString());
      double percentInfantryInOpen =
        Double.parseDouble(props.get("percentInfantryInOpen").toString());
      double percentInfantryDugIn =
        Double.parseDouble(props.get("percentInfantryDugIn").toString());
      double percentArmoredPC =
        Double.parseDouble(props.get("percentArmoredPC").toString());
      double percentLightSkinVehicle =
        Double.parseDouble(props.get("percentLightSkinVehicle").toString());

      int destroyArmor = Integer.parseInt(props.get("destroyArmor").toString());
      int destroyArmorSalvo =
Integer.parseInt(props.get("destroyArmorSalvo").toString());
      int neutralizeArmor = Integer.parseInt(props.get("neutralizeArmor").toString());
      int neutralizeArmorSalvo =
Integer.parseInt(props.get("neutralizeArmorSalvo").toString());
      int suppressArmor = Integer.parseInt(props.get("suppressArmor").toString());
      int suppressArmorSalvo =
Integer.parseInt(props.get("suppressArmorSalvo").toString());

      int destroyIIO = Integer.parseInt(props.get("destroyIIO").toString());
      int destroyIIOSalvo = Integer.parseInt(props.get("destroyIIOSalvo").toString());
      int neutralizeIIO = Integer.parseInt(props.get("neutralizeIIO").toString());
      int neutralizeIIOSalvo =
Integer.parseInt(props.get("neutralizeIIOSalvo").toString());
      int suppressIIO = Integer.parseInt(props.get("suppressIIO").toString());
      int suppressIIOSalvo = Integer.parseInt(props.get("suppressIIOSalvo").toString());

      int destroyIDI = Integer.parseInt(props.get("destroyIDI").toString());
      int destroyIDISalvo = Integer.parseInt(props.get("destroyIDISalvo").toString());
      int neutralizeIDI = Integer.parseInt(props.get("neutralizeIDI").toString());
      int neutralizeIDISalvo =
Integer.parseInt(props.get("neutralizeIDISalvo").toString());
      int suppressIDI = Integer.parseInt(props.get("suppressIDI").toString());
      int suppressIDISalvo = Integer.parseInt(props.get("suppressIDISalvo").toString());

      int destroyAPC = Integer.parseInt(props.get("destroyAPC").toString());
      int destroyAPCSalvo = Integer.parseInt(props.get("destroyAPCSalvo").toString());
      int neutralizeAPC = Integer.parseInt(props.get("neutralizeAPC").toString());
      int neutralizeAPCSalvo =
Integer.parseInt(props.get("neutralizeAPCSalvo").toString());
      int suppressAPC = Integer.parseInt(props.get("suppressAPC").toString());
      int suppressAPCSalvo = Integer.parseInt(props.get("suppressAPCSalvo").toString());

      int destroyLSV = Integer.parseInt(props.get("destroyLSV").toString());
      int destroyLSVSalvo = Integer.parseInt(props.get("destroyLSVSalvo").toString());
      int neutralizeLSV = Integer.parseInt(props.get("neutralizeLSV").toString());
      int neutralizeLSVSalvo =
Integer.parseInt(props.get("neutralizeLSVSalvo").toString());
      int suppressLSV = Integer.parseInt(props.get("suppressLSV").toString());
      int suppressLSVSalvo = Integer.parseInt(props.get("suppressLSVSalvo").toString());

      double percentDestroy = Double.parseDouble(props.get("percentDestroy").toString());
      double percentNeutralize =
Double.parseDouble(props.get("percentNeutralize").toString());
      double percentSuppress =
Double.parseDouble(props.get("percentSuppress").toString());

      // load BattleFieldData and shooters, instantiate the server
      BattleFieldData  battleFieldData = new BattleFieldData(percentArmor,destroyArmor,
        destroyArmorSalvo,neutralizeArmor,neutralizeArmorSalvo,suppressArmor,

suppressArmorSalvo,percentInfantryInOpen,destroyIIO,destroyIIOSalvo,neutralizeIIO,
        neutralizeIIOSalvo,suppressIIO,suppressIIOSalvo,percentInfantryDugIn,destroyIDI,
        destroyIDISalvo,neutralizeIDI,neutralizeIDISalvo,suppressIDI,suppressIDISalvo,
        percentArmoredPC,destroyAPC,destroyAPCSalvo,neutralizeAPC,neutralizeAPCSalvo,
        suppressAPC,suppressAPCSalvo,percentLightSkinVehicle,destroyLSV,destroyLSVSalvo,
        neutralizeLSV,neutralizeLSVSalvo,suppressLSV,suppressLSVSalvo,percentDestroy,
        percentNeutralize,percentSuppress,fleeProbability,
        RandomStream.STREAM[fleeStream],collateralDamageRadius,collateralDamagePercent);

      Vector theShooters = new Vector();
      for(int i=0;i<numberShooters;i++){
         theShooters.add(aShooter[i]);
      }

      double[] theWeights = new double[5];
      theWeights[0] = Double.parseDouble(props.get("range").toString());
      theWeights[1] = Double.parseDouble(props.get("thePER").toString());
      theWeights[2] = Double.parseDouble(props.get("numberRounds").toString());
      theWeights[3] = Double.parseDouble(props.get("FieldArtillery").toString());
      theWeights[4] = Double.parseDouble(props.get("NSFS").toString());

      TgtServer2 server = new TgtServer2(theShooters,box,battleFieldData,theWeights);

       SimpleStats fireMissionQueueTimeStat =
         new SimpleStats("fireMissionQueueTime",SamplingType.TALLY);
       SimpleStats totalFireMissionTimeStat =
         new SimpleStats("totalFireMissionTime",SamplingType.TALLY);
       SimpleStats tgtsInQueueStat =
         new SimpleStats("tgtsInQueue",SamplingType.TIME_VARYING);
       SimpleStats nasStat =
         new SimpleStats("numberAvailableShooters",SamplingType.TIME_VARYING);
       SimpleStats numberRepeatMissionsStat =
         new SimpleStats("numberMissionRepeats",SamplingType.TIME_VARYING);
       SimpleStats numberMissionsFiredStat =
         new SimpleStats("numberMissionsFired",SamplingType.TIME_VARYING);
       SimpleStats numberUnsuccessfulMissionsStat =
         new SimpleStats("numberUnsuccessfulMissions",SamplingType.TIME_VARYING);
       SimpleStats numberSuccessfulMissionsStat =
         new SimpleStats("numberSuccessfulMissions",SamplingType.TIME_VARYING);
       SimpleStats numberMissionsRejectedStat =
         new SimpleStats("numberMissionsRejected",SamplingType.TIME_VARYING);

       server.addPropertyChangeListener(fireMissionQueueTimeStat);
       server.addPropertyChangeListener(totalFireMissionTimeStat);
       server.addPropertyChangeListener(tgtsInQueueStat);
       server.addPropertyChangeListener(nasStat);
      server.addPropertyChangeListener(numberRepeatMissionsStat);
       server.addPropertyChangeListener(numberMissionsFiredStat);
       server.addPropertyChangeListener(numberUnsuccessfulMissionsStat);
       server.addPropertyChangeListener(numberSuccessfulMissionsStat);
       server.addPropertyChangeListener(numberMissionsRejectedStat);

       arrival.addSimEventListener(server);
          Schedule.stopOnEvent(stopEvent,stopEventCount);
       Schedule.setVerbose(false);
// FMT = Avg FM time
// NAS = Number available shooters
// PF = Percentage of missions fired
// TPS = Total percent of missions successful
// PS = Of missions fired, the percent of missions successful
// CD = Number of rds with collateral damage
// CA = percentage of area covered
// COA = the coa designator
      System.out.println(" FMT\t NAS\t PF\t TPS\t PS\t CD\t CA\t COA");
      for (int i=0;i<NUMREPLICATIONS;i++){
         Schedule.reset();
         fireMissionQueueTimeStat.reset();
         totalFireMissionTimeStat.reset();
         tgtsInQueueStat.reset();
         nasStat.reset();
         numberRepeatMissionsStat.reset();
         numberMissionsFiredStat.reset();
         numberUnsuccessfulMissionsStat.reset();
         numberSuccessfulMissionsStat.reset();
         numberMissionsRejectedStat.reset();
       Schedule.reset();
       Schedule.startSimulation();
         double fmt = totalFireMissionTimeStat.getMean();
         double nas = nasStat.getMean();
         double pf = (double)(arrival.getNumberArrivals()-
           numberMissionsRejectedStat.getCount())/arrival.getNumberArrivals();
         double tps = (double)numberSuccessfulMissionsStat.getCount()/
           (numberMissionsFiredStat.getCount()+numberMissionsRejectedStat.getCount());
         double ps = (double)numberSuccessfulMissionsStat.getCount()/
           numberMissionsFiredStat.getCount();
         int cd = server.getNumberRoundsCausingCollateralDamage();
         System.out.println(deci.format(fmt) + "\t" + deci.format(nas) + "\t" +
           deci.format(pf) + "\t" + deci.format(tps) + "\t" + deci.format(ps) + "\t  " +
           cd + "\t" + deci.format(ca) + "\t   " + theCOA);
      }
   }

}
```

2. FSSTGetCA.java

```
/**
 * Juan K. Ulloa <BR>
 * Thesis: FSST
 * January 19, 2001
 *
 * <P>
 *
**/

package fsst;
import simkit.*;
import simkit.data.*;
import simkit.smd.*;
import java.util.*;
import java.io.*;
import java.text.DecimalFormat;

public class FSSTGetCA{

// main method
   public static void main(String[] args){
      DecimalFormat deci = new DecimalFormat(" 0.0000;-0.0000");
        Properties props = new Properties();
           try {
               props.load(new FileInputStream(args[0]));
           }
           catch (FileNotFoundException e) {System.err.println(e);}
           catch (IOException e) {System.err.println(e);}

          // get COA data
        String theCOA = props.get("COA").toString();

          // get AO data
        double xLowerLeft = Double.parseDouble(props.get("xLowerLeft").toString());
        double yLowerLeft = Double.parseDouble(props.get("yLowerLeft").toString());
        double xUpperRight = Double.parseDouble(props.get("xUpperRight").toString());
        double yUpperRight= Double.parseDouble(props.get("yUpperRight").toString());
        String xDist = props.get("xDist").toString();
        double xDistA = Double.parseDouble(props.get("xDistA").toString());
        double xDistB = Double.parseDouble(props.get("xDistB").toString());
//      String yDist = props.get("yDist").toString();
//      double yDistMean = Double.parseDouble(props.get("yDistMean").toString());
        String yDist = props.get("yDist1").toString();
        double yDistA = Double.parseDouble(props.get("yDistA").toString());
        double yDistB = Double.parseDouble(props.get("yDistB").toString());

          // instantiate AO
      Coordinate lowerLeft = new Coordinate(xLowerLeft,yLowerLeft);
      Coordinate upperRight = new Coordinate(xUpperRight,yUpperRight);
      Object[] xDistParameters = new Object[]{new Double(xDistA),new Double(xDistB)};
//      Object[] yDistParameters = new Object[]{new Double(yDistMean)};
      Object[] yDistParameters = new Object[]{new Double(yDistA), new Double(yDistB)};
      AO box = new AO(lowerLeft,upperRight,xDist,xDistParameters,yDist,yDistParameters);

        // get the stop data
        String stopEvent = props.get("stopEvent").toString();
        int stopEventCount = Integer.parseInt(props.get("stopEventCount").toString());

        // get the ArrivalProcess data
        String interarrivalDistribution = props.get("arrivalDistribution").toString();
        double meanInterarrivalTime =
          Double.parseDouble(props.get("meanInterarrival").toString());
        int arrivalStream = Integer.parseInt(props.get("arrivalStream").toString());

        // instantiate the arrivalProcess
        TgtArrivalProcess arrival = new TgtArrivalProcess(interarrivalDistribution,
          new Object[] {new
Double(meanInterarrivalTime)},RandomStream.STREAM[arrivalStream]);

        // get the Shooter data
      int numberShooters = Integer.parseInt(props.get("numberShooters").toString());
        double meanProjectileVelocity =
         Double.parseDouble(props.get("meanProjectileVelocity").toString());
      String[] platformType = new String[numberShooters];
      String[] acquireServiceDistribution = new String[numberShooters];
      double[] meanAcquireInterservice = new double[numberShooters];
      double[] sigmaAcquire = new double[numberShooters];
      String[] firingServiceDistribution = new String[numberShooters];
      double[] meanFiringInterservice = new double[numberShooters];
      double[] sigmaFiring = new double[numberShooters];
      double[] maxRateOfFire = new double[numberShooters];
      double[] thePER = new double[numberShooters];
      double[] thePED = new double[numberShooters];
      double[] shooterXCoord = new double[numberShooters];
      double[] shooterYCoord = new double[numberShooters];
      int[] acquireServerStream = new int[numberShooters];
      int[] firingServerStream = new int[numberShooters];
      int[] numberRounds = new int[numberShooters];
      int[] numberGuns = new int[numberShooters];
      int[] shooterRange = new int[numberShooters];
      int[] burstRadius = new int[numberShooters];
      Shooter[] aShooter = new Shooter[numberShooters];

      for(int i=0;i<numberShooters;i++){
         platformType[i] = props.get("platformType"+(i+1)).toString();
        acquireServiceDistribution[i] =
           props.get("acquireServiceDistribution"+(i+1)).toString();
           meanAcquireInterservice[i] =
           Double.parseDouble(props.get("meanAcquireInterservice"+(i+1)).toString());
        sigmaAcquire[i] = Double.parseDouble(props.get("sigmaAcquire"+(i+1)).toString());
        firingServiceDistribution[i] =
           props.get("firingServiceDistribution"+(i+1)).toString();
           meanFiringInterservice[i] =
           Double.parseDouble(props.get("meanFiringInterservice"+(i+1)).toString());
        sigmaFiring[i] = Double.parseDouble(props.get("sigmaFiring"+(i+1)).toString());
         maxRateOfFire[i] =
           Double.parseDouble(props.get("maxRateOfFire"+(i+1)).toString());
        thePER[i] = Double.parseDouble(props.get("thePER"+(i+1)).toString());
         thePED[i] = Double.parseDouble(props.get("thePED"+(i+1)).toString());
         shooterXCoord[i] =
Double.parseDouble(props.get("shooterXCoord"+(i+1)).toString());
         shooterYCoord[i] =
Double.parseDouble(props.get("shooterYCoord"+(i+1)).toString());
        acquireServerStream[i] =
           Integer.parseInt(props.get("acquireServerStream"+(i+1)).toString());
        firingServerStream[i] =
           Integer.parseInt(props.get("firingServerStream"+(i+1)).toString());
        numberRounds[i] = Integer.parseInt(props.get("numberRounds"+(i+1)).toString());
        numberGuns[i] = Integer.parseInt(props.get("numberGuns"+(i+1)).toString());
         shooterRange[i] = Integer.parseInt(props.get("shooterRange"+(i+1)).toString());
         burstRadius[i] = Integer.parseInt(props.get("burstRadius"+(i+1)).toString());
            // instantiate the shooters
         aShooter[i] = new Shooter(new Coordinate(shooterXCoord[i],shooterYCoord[i]),
           shooterRange[i],thePER[i],thePED[i],acquireServiceDistribution[i],
           new Object[] {new Double(meanAcquireInterservice[i]),
           new Double(sigmaAcquire[i])},firingServiceDistribution[i],new Object[]
           {new Double(meanFiringInterservice[i]),new Double(sigmaFiring[i])},
           maxRateOfFire[i],RandomStream.STREAM[firingServerStream[i]],
           RandomStream.STREAM[acquireServerStream[i]],numberRounds[i],
           burstRadius[i],meanProjectileVelocity, platformType[i],numberGuns[i]);
      }

         // instantiate the BattleFieldData
      double fleeProbability =
Double.parseDouble(props.get("fleeProbability").toString());
      int fleeStream = Integer.parseInt(props.get("fleeStream").toString());
      double collateralDamageRadius =
        Double.parseDouble(props.get("collateralDamageRadius").toString());
      double collateralDamagePercent =
        Double.parseDouble(props.get("collateralDamagePercent").toString());
      double percentArmor =
        Double.parseDouble(props.get("percentArmor").toString());
      double percentInfantryInOpen =
        Double.parseDouble(props.get("percentInfantryInOpen").toString());
      double percentInfantryDugIn =
        Double.parseDouble(props.get("percentInfantryDugIn").toString());
      double percentArmoredPC =
        Double.parseDouble(props.get("percentArmoredPC").toString());
      double percentLightSkinVehicle =
        Double.parseDouble(props.get("percentLightSkinVehicle").toString());

      int destroyArmor = Integer.parseInt(props.get("destroyArmor").toString());
      int destroyArmorSalvo =
Integer.parseInt(props.get("destroyArmorSalvo").toString());
      int neutralizeArmor = Integer.parseInt(props.get("neutralizeArmor").toString());
      int neutralizeArmorSalvo =
Integer.parseInt(props.get("neutralizeArmorSalvo").toString());
      int suppressArmor = Integer.parseInt(props.get("suppressArmor").toString());
      int suppressArmorSalvo =
Integer.parseInt(props.get("suppressArmorSalvo").toString());

      int destroyIIO = Integer.parseInt(props.get("destroyIIO").toString());
      int destroyIIOSalvo = Integer.parseInt(props.get("destroyIIOSalvo").toString());
      int neutralizeIIO = Integer.parseInt(props.get("neutralizeIIO").toString());
      int neutralizeIIOSalvo =
Integer.parseInt(props.get("neutralizeIIOSalvo").toString());
      int suppressIIO = Integer.parseInt(props.get("suppressIIO").toString());
      int suppressIIOSalvo = Integer.parseInt(props.get("suppressIIOSalvo").toString());

      int destroyIDI = Integer.parseInt(props.get("destroyIDI").toString());
      int destroyIDISalvo = Integer.parseInt(props.get("destroyIDISalvo").toString());
      int neutralizeIDI = Integer.parseInt(props.get("neutralizeIDI").toString());
      int neutralizeIDISalvo =
Integer.parseInt(props.get("neutralizeIDISalvo").toString());
      int suppressIDI = Integer.parseInt(props.get("suppressIDI").toString());
      int suppressIDISalvo = Integer.parseInt(props.get("suppressIDISalvo").toString());

      int destroyAPC = Integer.parseInt(props.get("destroyAPC").toString());
      int destroyAPCSalvo = Integer.parseInt(props.get("destroyAPCSalvo").toString());
      int neutralizeAPC = Integer.parseInt(props.get("neutralizeAPC").toString());
      int neutralizeAPCSalvo =
Integer.parseInt(props.get("neutralizeAPCSalvo").toString());
      int suppressAPC = Integer.parseInt(props.get("suppressAPC").toString());
      int suppressAPCSalvo = Integer.parseInt(props.get("suppressAPCSalvo").toString());

      int destroyLSV = Integer.parseInt(props.get("destroyLSV").toString());
      int destroyLSVSalvo = Integer.parseInt(props.get("destroyLSVSalvo").toString());
      int neutralizeLSV = Integer.parseInt(props.get("neutralizeLSV").toString());
      int neutralizeLSVSalvo =
Integer.parseInt(props.get("neutralizeLSVSalvo").toString());
      int suppressLSV = Integer.parseInt(props.get("suppressLSV").toString());
      int suppressLSVSalvo = Integer.parseInt(props.get("suppressLSVSalvo").toString());

      double percentDestroy = Double.parseDouble(props.get("percentDestroy").toString());
      double percentNeutralize =
Double.parseDouble(props.get("percentNeutralize").toString());
      double percentSuppress =
Double.parseDouble(props.get("percentSuppress").toString());

      // load BattleFieldData and shooters, instantiate the server
      BattleFieldData  battleFieldData = new BattleFieldData(percentArmor,destroyArmor,
        destroyArmorSalvo,neutralizeArmor,neutralizeArmorSalvo,suppressArmor,

suppressArmorSalvo,percentInfantryInOpen,destroyIIO,destroyIIOSalvo,neutralizeIIO,
        neutralizeIIOSalvo,suppressIIO,suppressIIOSalvo,percentInfantryDugIn,destroyIDI,
        destroyIDISalvo,neutralizeIDI,neutralizeIDISalvo,suppressIDI,suppressIDISalvo,
        percentArmoredPC,destroyAPC,destroyAPCSalvo,neutralizeAPC,neutralizeAPCSalvo,
        suppressAPC,suppressAPCSalvo,percentLightSkinVehicle,destroyLSV,destroyLSVSalvo,
        neutralizeLSV,neutralizeLSVSalvo,suppressLSV,suppressLSVSalvo,percentDestroy,
        percentNeutralize,percentSuppress,fleeProbability,
        RandomStream.STREAM[fleeStream],collateralDamageRadius,collateralDamagePercent);

      Vector theShooters = new Vector();
      for(int i=0;i<numberShooters;i++){
         theShooters.add(aShooter[i]);
      }

      double[] theWeights = new double[5];
      theWeights[0] = Double.parseDouble(props.get("range").toString());
      theWeights[1] = Double.parseDouble(props.get("thePER").toString());
      theWeights[2] = Double.parseDouble(props.get("numberRounds").toString());
      theWeights[3] = Double.parseDouble(props.get("FieldArtillery").toString());
      theWeights[4] = Double.parseDouble(props.get("NSFS").toString());

      TgtServer2 server = new TgtServer2(theShooters,box,battleFieldData,theWeights);

       SimpleStats numberMissionsFiredStat =
         new SimpleStats("numberMissionsFired",SamplingType.TIME_VARYING);
       SimpleStats numberMissionsRejectedStat =
         new SimpleStats("numberMissionsRejected",SamplingType.TIME_VARYING);

       server.addPropertyChangeListener(numberMissionsFiredStat);
       server.addPropertyChangeListener(numberMissionsRejectedStat);

       arrival.addSimEventListener(server);
          Schedule.stopOnEvent(stopEvent,stopEventCount);
       Schedule.setVerbose(false);
// theCA = percentage of area covered
      Schedule.reset();
      numberMissionsFiredStat.reset();
      numberMissionsRejectedStat.reset();
       Schedule.reset();
       Schedule.startSimulation();
      double theCA = (double)numberMissionsFiredStat.getCount()/
        (numberMissionsFiredStat.getCount()+numberMissionsRejectedStat.getCount());
      System.out.println("The COA = " + theCOA + "\nThe CA = " + deci.format(theCA));

      for(int i=0;i<numberShooters;i++){
         System.out.println(aShooter[i].toString());
      }
   }

}
```

C. PROPERTIES FILE EXAMPLES

1. IBCT: Army with Reinforcing NSFS

```
# Fire Support Simulation Tool
# 12 January 2001
#
# File: fsst1.properties
# Properties file for Multiple Server Queueing Model with Reneging
#

# The COA Designator

COA = 1D
theCA = 1.0

# The AO properties
xLowerLeft = 0.0
yLowerLeft = 0.0
xUpperRight = 24000.0
yUpperRight = 70000.0
xDist = Uniform
xDistA = 0.0
xDistB = 24000.0
yDist = Exponential
yDistMean = 50000
yDist1 = Uniform
yDistA = 0.0
yDistB = 70000.0


# ArrivalProcess properties
arrivalDistribution = Exponential
meanInterarrival = 2.5
arrivalStream = 1

# BattleFieldData
fleeProbability = 0.9
fleeStream = 2
collateralDamageRadius = 400
collateralDamagePercent = 0.01

   # Probability of Target Type being:
percentArmor = 0.4
percentInfantryInOpen = 0.1
percentInfantryDugIn = 0.0
percentArmoredPC = 0.3
percentLightSkinVehicle = 0.2

   # Number of Rounds within Burst Radius of Target to Destroy:
destroyArmor = 18
destroyIIO = 3
destroyIDI = 6
destroyAPC = 12
destroyLSV = 9

   # Number of Rounds within Burst Radius of Target to Neutralize:
neutralizeArmor = 12
neutralizeIIO = 2
neutralizeIDI = 4
neutralizeAPC = 9
neutralizeLSV = 6

   # Number of Rounds within 2 * (Burst Radius) of Target to Suppress:
suppressArmor = 1
suppressIIO = 1
suppressIDI = 1
suppressAPC = 1
```

```
suppressLSV = 1

   # Number of rounds to fire in a salvo against specific targets:
destroyArmorSalvo = 36
neutralizeArmorSalvo = 24
suppressArmorSalvo = 3

destroyIIOSalvo = 6
neutralizeIIOSalvo = 3
suppressIIOSalvo = 3

destroyIDISalvo = 24
neutralizeIDISalvo = 9
suppressIDISalvo = 3

destroyAPCSalvo = 24
neutralizeAPCSalvo = 18
suppressAPCSalvo = 3

destroyLSVSalvo = 18
neutralizeLSVSalvo = 12
suppressLSVSalvo = 3

   # Probability of mission being:
percentDestroy = 0.3
percentNeutralize = 0.5
percentSuppress = 0.2

# Shooter properties
        # projo velocity in x direction in m/min ( (mvv*60)/sqrt(2) )
meanProjectileVelocity = 70000
numberShooters = 6

# Shooter1 properties (Paladin Battery)
platformType1 = FieldArtillery
acquireServiceDistribution1 = Normal
meanAcquireInterservice1 = 4.0
sigmaAcquire1 = 0.75
firingServiceDistribution1 = Normal
meanFiringInterservice1  = 1.0
sigmaFiring1 = 0.2
 # in rounds per minute
maxRateOfFire1 = 8
thePER1 = 35
thePED1 = 2
shooterXCoord1 = 6000.0
shooterYCoord1 = 30000.0
acquireServerStream1 = 3
firingServerStream1 = 4
numberRounds1 = 2520
shooterRange1 = 30000
burstRadius1 = 50
numberGuns1 = 6

# Shooter2 properties (Paladin Battery)
platformType2 = FieldArtillery
acquireServiceDistribution2 = Normal
meanAcquireInterservice2 = 4.0
sigmaAcquire2 = 0.75
firingServiceDistribution2 = Normal
meanFiringInterservice2  = 1.0
sigmaFiring2 = 0.2
 # in rounds per minute
maxRateOfFire2 = 8
thePER2 = 35
thePED2 = 2
shooterXCoord2 = 12000.0
shooterYCoord2 = 30000.0
acquireServerStream2 = 5
firingServerStream2 = 6
numberRounds2 = 2520
```

```
shooterRange2 = 30000
burstRadius2 = 50
numberGuns2 = 6

# Shooter3 properties (Paladin Battery)
platformType3 = FieldArtillery
acquireServiceDistribution3 = Normal
meanAcquireInterservice3 = 4.0
sigmaAcquire3 = 0.75
firingServiceDistribution3 = Normal
meanFiringInterservice3  = 1.0
sigmaFiring3 = 0.2
 # in rounds per minute
maxRateOfFire3 = 8
thePER3 = 35
thePED3 = 2
shooterXCoord3 = 18000.0
shooterYCoord3 = 30000.0
acquireServerStream3 = 7
firingServerStream3 = 8
numberRounds3 = 2520
shooterRange3 = 30000
burstRadius3 = 50
numberGuns3 = 6

# Shooter4 properties
platformType4 = NSFS
acquireServiceDistribution4 = Normal
meanAcquireInterservice4 = 5.0
sigmaAcquire4 = 1.0
firingServiceDistribution4 = Normal
meanFiringInterservice4  = 0.5
sigmaFiring4 = 0.1
   # in rounds per minute
maxRateOfFire4 = 24
thePER4 = 100
thePED4 = 2
shooterXCoord4 = 6000.0
shooterYCoord4 = -40000.0
acquireServerStream4 = 9
firingServerStream4 = 0
numberRounds4 = 2400
shooterRange4 = 112000
burstRadius4 = 75
numberGuns4 = 2

# Shooter5 properties
platformType5 = NSFS
acquireServiceDistribution5 = Normal
meanAcquireInterservice5 = 5.0
sigmaAcquire5 = 1.0
firingServiceDistribution5 = Normal
meanFiringInterservice5  = 0.5
sigmaFiring5 = 0.1
   # in rounds per minute
maxRateOfFire5 = 24
thePER5 = 100
thePED5 = 2
shooterXCoord5 = 12000.0
shooterYCoord5 = -40000.0
acquireServerStream5 = 1
firingServerStream5 = 2
numberRounds5 = 2400
shooterRange5 = 112000
burstRadius5 = 75
numberGuns5 = 2

# Shooter6 properties
platformType6 = NSFS
acquireServiceDistribution6 = Normal
meanAcquireInterservice6 = 5.0
```

```
sigmaAcquire6 = 1.0
firingServiceDistribution6 = Normal
meanFiringInterservice6  = 0.5
sigmaFiring6 = 0.1
   # in rounds per minute
maxRateOfFire6 = 24
thePER6 = 100
thePED6 = 2
shooterXCoord6 = 18000.0
shooterYCoord6 = -40000.0
acquireServerStream6 = 4
firingServerStream6 = 5
numberRounds6 = 2400
shooterRange6 = 112000
burstRadius6 = 75
numberGuns6 = 2

# the Weights
range = 0.125
thePER = 0.125
numberRounds = .5
FieldArtillery = 0.25
NSFS = 0.00

# Run execute properties
stopEvent = TargetArrival
stopEventCount = 425
```

2. FCS: Army with Reinforcing NSFS

```
# Fire Support Simulation Tool
# 12 January 2001
#
# File: fsst1.properties
# Properties file for Multiple Server Queueing Model with Reneging
#

# The COA Designator

COA = 2D
theCA = 1.0

# The AO properties
xLowerLeft = 0.0
yLowerLeft = 0.0
xUpperRight = 50000.0
yUpperRight = 120000.0
xDist = Uniform
xDistA = 0.0
xDistB = 50000.0
yDist = Exponential
yDistMean = 80000
yDist1 = Uniform
yDistA = 0.0
yDistB = 120000.0


# ArrivalProcess properties
arrivalDistribution = Exponential
meanInterarrival = 2.5
arrivalStream = 1

# BattleFieldData
fleeProbability = 0.9
fleeStream = 2
collateralDamageRadius = 400
collateralDamagePercent = 0.01

# Probability of Target Type being:
```

```
percentArmor = 0.4
percentInfantryInOpen = 0.1
percentInfantryDugIn = 0.0
percentArmoredPC = 0.3
percentLightSkinVehicle = 0.2

   # Number of Rounds within Burst Radius of Target to Destroy:
destroyArmor = 18
destroyIIO = 3
destroyIDI = 6
destroyAPC = 12
destroyLSV = 9

   # Number of Rounds within Burst Radius of Target to Neutralize:
neutralizeArmor = 12
neutralizeIIO = 2
neutralizeIDI = 4
neutralizeAPC = 9
neutralizeLSV = 6

   # Number of Rounds within 2 * (Burst Radius) of Target to Suppress:
suppressArmor = 1
suppressIIO = 1
suppressIDI = 1
suppressAPC = 1
suppressLSV = 1

   # Number of rounds to fire in a salvo against specific targets:
destroyArmorSalvo = 18
neutralizeArmorSalvo = 12
suppressArmorSalvo = 3

destroyIIOSalvo = 3
neutralizeIIOSalvo = 3
suppressIIOSalvo = 3

destroyIDISalvo = 6
neutralizeIDISalvo = 4
suppressIDISalvo = 3

destroyAPCSalvo = 12
neutralizeAPCSalvo = 9
suppressAPCSalvo = 3

destroyLSVSalvo = 9
neutralizeLSVSalvo = 6
suppressLSVSalvo = 3

   # Probability of mission being:
percentDestroy = 0.3
percentNeutralize = 0.5
percentSuppress = 0.2

# Shooter properties
       # projo velocity in x direction in m/min ( (mvv*60)/sqrt(2) )
meanProjectileVelocity = 100000
numberShooters = 6

# Shooter1 properties (Crusader Battery)
platformType1 = FieldArtillery
acquireServiceDistribution1 = Normal
meanAcquireInterservice1 = 2.0
sigmaAcquire1 = 0.25
firingServiceDistribution1 = Normal
meanFiringInterservice1  = 0.25
sigmaFiring1 = 0.05
 # in rounds per minute
maxRateOfFire1 = 12
thePER1 = 1
thePED1 = 1
shooterXCoord1 = 12500.0
```

```
shooterYCoord1 = 70000.0
acquireServerStream1 = 3
firingServerStream1 = 4
numberRounds1 = 2520
shooterRange1 = 45000
burstRadius1 = 50
numberGuns1 = 4

# Shooter2 properties (Crusader Battery)
platformType2 = FieldArtillery
acquireServiceDistribution2 = Normal
meanAcquireInterservice2 = 2.0
sigmaAcquire2 = 0.25
firingServiceDistribution2 = Normal
meanFiringInterservice2  = 0.25
sigmaFiring2 = 0.05
 # in rounds per minute
maxRateOfFire2 = 12
thePER2 = 1
thePED2 = 1
shooterXCoord2 = 25000.0
shooterYCoord2 = 70000.0
acquireServerStream2 = 5
firingServerStream2 = 6
numberRounds2 = 2520
shooterRange2 = 45000
burstRadius2 = 50
numberGuns2 = 4

# Shooter3 properties (Crusader Battery)
platformType3 = FieldArtillery
acquireServiceDistribution3 = Normal
meanAcquireInterservice3 = 2.0
sigmaAcquire3 = 0.25
firingServiceDistribution3 = Normal
meanFiringInterservice3  = 0.25
sigmaFiring3 = 0.05
 # in rounds per minute
maxRateOfFire3 = 12
thePER3 = 1
thePED3 = 1
shooterXCoord3 = 37500.0
shooterYCoord3 = 70000.0
acquireServerStream3 = 7
firingServerStream3 = 8
numberRounds3 = 2520
shooterRange3 = 45000
burstRadius3 = 50
numberGuns3 = 4

# Shooter4 properties
platformType4 = NSFS
acquireServiceDistribution4 = Normal
meanAcquireInterservice4 = 2.5
sigmaAcquire4 = 0.5
firingServiceDistribution4 = Normal
meanFiringInterservice4  = 0.5
sigmaFiring4 = 0.1
   # in rounds per minute
maxRateOfFire4 = 24
thePER4 = 10
thePED4 = 2
shooterXCoord4 = 6000.0
shooterYCoord4 = -40000.0
acquireServerStream4 = 9
firingServerStream4 = 0
numberRounds4 = 2400
shooterRange4 = 162000
burstRadius4 = 75
numberGuns4 = 2
```

```
# Shooter5 properties
platformType5 = NSFS
acquireServiceDistribution5 = Normal
meanAcquireInterservice5 = 2.5
sigmaAcquire5 = 0.5
firingServiceDistribution5 = Normal
meanFiringInterservice5  = 0.5
sigmaFiring5 = 0.1
   # in rounds per minute
maxRateOfFire5 = 24
thePER5 = 10
thePED5 = 2
shooterXCoord5 = 12000.0
shooterYCoord5 = -40000.0
acquireServerStream5 = 1
firingServerStream5 = 2
numberRounds5 = 2400
shooterRange5 = 162000
burstRadius5 = 75
numberGuns5 = 2

# Shooter6 properties
platformType6 = NSFS
acquireServiceDistribution6 = Normal
meanAcquireInterservice6 = 2.5
sigmaAcquire6 = 0.5
firingServiceDistribution6 = Normal
meanFiringInterservice6  = 0.5
sigmaFiring6 = 0.1
   # in rounds per minute
maxRateOfFire6 = 24
thePER6 = 10
thePED6 = 2
shooterXCoord6 = 18000.0
shooterYCoord6 = -40000.0
acquireServerStream6 = 4
firingServerStream6 = 5
numberRounds6 = 2400
shooterRange6 = 162000
burstRadius6 = 75
numberGuns6 = 2

# the Weights
range = 0.125
thePER = 0.125
numberRounds = .5
FieldArtillery = 0.25
NSFS = 0.00

# Run execute properties
stopEvent = TargetArrival
stopEventCount = 425
```

THIS PAGE INTENTIONALLY LEFT BLANK

BIBLIOGRAPHY

FM 100-5, “Operations,” Headquarters, Department of the Army, Washington, D.C., 14 June 1993.

United States Army, *Weapon Systems, United States Army 1999,* DIANA Publishing Company, 1999.

Johns Hopkins University/Applied Physics Lab, Task Statement for NSFS Requirements for US Army Future Combat System Objective Force, 08 March 2000, JHU/APL Proprietary.

J.E. Rhodes - United States Marine Corps, Memorandum to the Chief of Naval Operations from the United States Marine Corps on Naval Surface Fire Support Requirements for Operational Maneuver from the Sea – 1999, 16 June 1999.

Sean D. Naylor, “Next up: Looking beyond Force XXI Army’s newest project takes tactical look at warfare in the years between 2010 and 2025,” *Army Times,* 10 June 1996, p. 18.

Brigadier General Edward T. Buckley Jr. – US Army, “Army After Next Technology: Forging Possibilities into Reality,” available at http://www-cgsc-army.mil/milrev/English/MarApr98/buckley.htm; Internet; accessed 21 November 2000.

Colonel Michael Mehaffey – US Army, “Vanguard of the Objective Force,” available at http://www-cgsc-army.mil/milrev/English/SepOct00/meha.htm; Internet; accessed 15 November 2000.

Major General Joseph M. Cosumano Jr. – US Army, “Transforming the Army to a Full Spectrum Force,” available at http://www.tradoc.army.mil/transformation…ges/ transforming_the_army_to_a_full_.htm; Internet; accessed 20 November 2000.

Army Chief of Staff General Eric K. Shinseki, “Army Vision address at Reserve Officers Association Mid-Winter Conference 1999,” available at http://jwadweb/abig/5.htm; JHU Intranet, accessed 15 November 2000.

Army Chief of Staff General Eric K. Shinseki, “Statement on Army Readiness to 106th Congress,” available at http://gopher .house.gov/hasc/testimony/106thcongress/00-02-10shinseki.htm; Internet; accessed 20 November 2000.

Army Chief of Staff General Eric K. Shinseki, “Statement of Army Vision,” available at http://www.senate.gov/~appropriations/defense/testimony/shinseki.htm; Internet; accessed 20 November 2000.

Crusader 155MM, The WebSite for Defence Industries – Army, available at http://www.army-technology.com/projects/crusader/

Paladin 155MM, The WebSite for Defence Industries – Army, available at http://www.army-technology.com/projects/paladin/
A.D. Zimm, R.P. O'Neil, D.W. Kerchner, and E.A. Smith, "Draft - Naval Requirements and Capabilities Study," 13 January 2000, Johns Hopkins University Applied Physics Lab., pp. 65-72.

Raymond Lisiewski and Edward C. Whitmann, "DD21: A New Direction in Warship Acquisition," Paper on genesis of DD-21 operational requirements, May 2000, available at http://dd21.crane.navy.mil/pdf%20files/DD21Whitman.pdf, accessed 20 November 2000.

Le, Hung B., "Advanced Naval Surface Fire Support Weapon Employment Against Mobile Targets." Masters Thesis, Naval Postgraduate School, December 1999.

Armstrong, James E., SE401/SE402, *Introduction to Systems Design, Course Notes*, West Point, New York: Department of Systems Engineering, 1999.

Averill M. Law and W. David Kelton, *Simulation Modeling and Analysis*, 3d ed., (Boston: McGraw-Hill, 2000).

G.E.P. Box, W.G. Hunter, and J.S. Hunter, *Statistics for Experimenters, An Introduction to Design, Data Analysis, and Model Building*, (New York: John Wiley and Sons, 1978).

Jay L. Devore, *Probability and Statistics for Engineering and the Sciences*, 4th ed., (Pacific Grove: Duxbury Press, 1995).

Douglas C. Montgomery, George C. Runger, and Norma F. Hubele, *Engineering Statistics*, (New York: John Wiley and Sons, 1998).

John R. Canada and William G. Sullivan, *Economic Multiattribute Evaluation of Advanced Manufacturing Systems*, (Englewood Cliffs: Prentice Hall, 1989).

James N. Eagle, II, Radially Fleeing Target Animation in MATLAB, available at: http://spica.or.nps.navy.mil/searchdocs/demos/rad_flee_norm.ani.gif, accessed 25 April 2001.

Stork, Kurt A., "Sensors in Object Oriented Discrete Event Simulation." Masters Thesis, Naval Postgraduate School, September 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library...2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Mr. Edward A. Smyth...1
   Joint Warfare Analysis Department, JHU/APL
   Johns Hopkins Road
   Laurel, Maryland 20723-6099

4. Mr. John F. Keane..1
   Joint Warfare Analysis Department, JHU/APL
   Johns Hopkins Road
   Laurel, Maryland 20723-6099

5. Mr. Stephen M. Orloff 1
   Joint Warfare Analysis Department, JHU/APL
   Johns Hopkins Road
   Laurel, Maryland 20723-6099

6. Lieutenant Colonel Eugene P. Paulo, Pe 2
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California 93943-5000

7. Professor Arnold H. Buss. Bu...2
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California 93943-5000

8. Major Juan K. Ulloa...2
   113 Wildplum Road
   Lawton, OK 73501